When you know that you're capable of dealing with
whatever comes, you have the only security the world has to offer.
Harry Browne

University of Alberta

Real Time Threat Mitigation Techniques

by

Joshua Dustin Ryder

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2006

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-22363-5
Our file Notre référence
ISBN: 978-0-494-22363-5

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Computer networks are constantly the victims of unwanted poking, probes, and outright violations. As the tools available to potential attackers have increased in sophistication, so has the need of counter measures to prevent such unwanted advances. Honeypots are emerging as a particularly interesting defense mechanism. Honeypots are software constructs that provide the ability to create the illusion of a computer system existing where there is none. These illusions simulate a computer system by recreating some of that system's signature interaction mechanisms. By definition, because these systems do not exist, any traffic being sent to them should be viewed with suspicion.

A novel integration between honeypots and firewalls, called a honeywall, has proven to be an effective tool in hiding a production network environment from prying eyes within a facade of false computer systems. The honeywall increases the response-time window available to computer security experts. This provides the attacked site the option of passively monitoring the intruder's activities, routing the intruder to a completely fictitious network residing on the honeywall itself, or actively responding to the threat while building an attacker profile.

This thesis demonstrates how a honeywall can be successfully used to mitigate rapidly spreading Internet worms in real-time.

Acknowledgements

The process experienced by me during the creation of this dissertation is, I suspect, not unlike that which countless researchers before me have likewise undergone. That is to say, this dissertation would most likely have never seen the light of day had it not been for several key people's contributions along the way. When I look back on this document years down the road, it will be not only with pride, but also with the flood of memories from events that took place during the writing. Please take a moment with me to recognize them.

Donna Gorday and Jason Clements of the AICT Labs Group, who generously provided what became the testing facilities used in this research.
Kevin Watts and Raymond Richmond of AICT Network Operations: Cisco expertise.
Bob Beck for his motivating speeches and being an excellent sounding board.
Chris Kuethe for his OpenBSD and gnuplot wizardry, for always making time to listen to and discuss ideas, and for his unwavering friendship.
Lance Spitzner for our chat at pacsec.jp '03.
Jenney McNaughton for her Adobe expertise and amazing ability to take my scratchings and turn them into printable diagrams.
Dragos Ruiu for bringing together such an amazing group of individuals for pacsec.jp '03.
Ron Unrau: patience and deadlines.
Jonathan Schaeffer: fear.
Amanda Lorier, or more properly now, Dr. Amanda Lorier. I can't even begin to put into words how she has changed my life for the better, to thank her enough for her love and support, or to repay the hugs and smiles that kept me going along the way.
My family, both here and gone. To my Opa and Grandma, I wish you could see me now. Grandad, for all the life lessons you provided when I was growing up, and the ones you keep sharing now. Mom, you were right. Dad, thanks. Jeremy, looking forward to reading your dissertation in a few years. Kristy, don't ever change kiddo.

Contents

1 Introduction **1**
- 1.1 Motivation and Scope . . . 1
- 1.2 Overview of Threat . . . 2
  - 1.2.1 Examination of a Virus . . . 2
  - 1.2.2 Examination of a Worm . . . 4
  - 1.2.3 Honeypots and Honeywalls . . . 4
- 1.3 Overview of the Research . . . 5
- 1.4 Contributions . . . 7
- 1.5 Overview of Thesis . . . 7

2 Background **9**
- 2.1 Network Activity . . . 9
- 2.2 Types of Attacks . . . 10
  - 2.2.1 Denial of Service Attacks . . . 11
  - 2.2.2 Exploits . . . 12
  - 2.2.3 Buffer Overflows . . . 13
- 2.3 Firewalls . . . 14
- 2.4 Intrusion Detection Systems . . . 17
- 2.5 Honeypots . . . 19
  - 2.5.1 Low Interaction . . . 19
  - 2.5.2 High Interaction . . . 20
  - 2.5.3 Honeypots in Practice . . . 20
  - 2.5.4 Sniffing . . . 21
  - 2.5.5 Scanning . . . 22
  - 2.5.6 Fingerprinting . . . 24

3 State of the Art and Design **25**
- 3.1 Firewalls . . . 25
  - 3.1.1 Border Firewalls . . . 26
  - 3.1.2 Host-Based Firewalls . . . 27
  - 3.1.3 Static Firewalls . . . 30
  - 3.1.4 Reactive Firewalls . . . 31
  - 3.1.5 Intrusion Detection Systems . . . 32
  - 3.1.6 pf . . . 33
- 3.2 Honeypots and Honeynets . . . 34
- 3.3 Honeywalls . . . 35

4 Evaluation **37**
4.1 Test Harness 38
4.2 Performance Indicators 40
4.2.1 Number of Systems Compromised 40
4.2.2 Time to Stable State 40
4.3 Test Worm Configuration 41
4.4 Lab Configuration 44
4.4.1 Client Machine Hardware 44
4.4.2 Protection Hardware 44
4.4.3 VPN tunnels 45
4.5 Methodology 45
4.6 Static Firewall 45
4.7 Reactive Firewall 48
4.7.1 IDS Using pf 48
4.7.2 IDS Using Snort 53
4.8 Honeywall 56
4.9 Summary of Findings 58

5 Discussion **61**

6 Conclusion and Future Work **63**
6.1 False Positive Reduction 63
6.2 Honeypot Sensor Placement 64
6.3 Integration of Solutions 65
6.4 pf improvements 65
6.5 Response Times 66

Bibliography **67**

Chapter 1

Introduction

1.1 Motivation and Scope

The task of protecting networked assets has fallen to the network security professional. Unfortunately the tools available to detect and prevent intrusions have not kept pace with the attacker's tools to intrude. As a result, the ability to contain and mitigate potential network threats have been limited in effectiveness.

The goal of this research is to explore methods available to perform intrusion prevention and threat mitigation. Specifically, we are interested in addressing the ongoing problem area of so-called zero-day exploits in the guise of mass spreading worms. A zero-day worm takes advantage of computer systems that, while perhaps previously having been deemed secured, have been discovered to run software vulnerable to the payload delivered by the worm. In some cases these vulnerabilities are exploited before the maintainers of the vulnerable program are aware that their software has been compromised, and therefore have not been able to address the vulnerability via a patch or an update to their software. In the instance where the vulnerable software has been widely deployed, the potential amount of disruption and damage done by a rapidly spreading worm can be tremendous.

In 2003 a mass spreading e-mail worm labeled SoBig was released onto the Internet. Tens of thousands of vulnerable computer systems were infected by SoBig. Thousands more were disrupted in varying degrees of severity by the enormous load placed on the network as the infected machines attempted to spread the worm farther. Taking into account downtime, lost business, the cost of investing in new

technology and wasted productivity, estimates of SoBig's impact range from 500 million to 2 billion dollars [1].

The initial motivation for this thesis evolved from a perceived deficiency in existing mechanisms to address containment and mitigation of the next rapidly spreading worm within a heterogeneous distributed and compartmentalized network, as characterized by that of the University of Alberta. It is desired that the thesis be viewed as a methodological inspection of existing techniques deployed in similar environments.

1.2 Overview of Threat

There are several different methods available to disrupt the normal function of a computer network. For example, a simple high-rate packet flood from a sufficient number of sources is often enough to affect or disrupt a computer network. Another example would be a directed attack where an individual targets a set of specific systems on a network and attempts to gain unauthorized access to them either through direct compromise, or through other social and/or technical means. For the purposes of this dissertation we will be examining techniques that are available today for addressing the problem of rapid worm propagation across the vulnerable computer systems on a network, as well as including results from testing a novel mechanism developed for the express purpose of detecting and isolating fast spreading worms.

As there are many differing methods of classifying worms and viruses, we include below a brief discussion of each and how they apply to the problem that is being examined.

1.2.1 Examination of a Virus

Cohen defines a computer virus to be "a computer program that can affect other computer programs by modifying them in such a way as to include a (possibly evolved) copy of itself" [2]. Computer viruses have been present in one form or another for decades. In 1981, in answer to a debate regarding the piracy of computer games exhibiting traits of evolution and natural selection at Texas A&M, an

anonymous individual created the first virus on record [3]. This program, called Elk Cloner, was designed to integrate with the Apple II operating system present on that platform's standard formatted diskettes [4]. When a specific command was entered by the user on a system using an infected floppy, the virus would transfer itself to all other floppies placed in that system. Among other mischief, every 50th reboot of the system a message would appear containing a poem about the Elk Cloner. The idea of having a virus display text to the user remains a common theme among virus writers even today.

Early computer viruses spread slowly by today's standards, hampered by the latency introduced by their chosen spreading mechanisms. For example, a virus which writes itself to the bootsector of a floppy disk, such as MONKEY.A or STONED, is limited in impact by the visibility and transportability of the media [5]. If an infected floppy is only used on one computer system and that computer's user never shares diskettes used on that system, then the virus can't spread. Despite this relatively slow transport mechanism, in 1991 – the same year Linux 0.01 was introduced by Linus Torvalds and less than one year after Berners-Lee and Cailliau released hypertext to the Internet – the Monkey bootsector virus was able to spread globally [6] [7].

As certain programs became increasingly ubiquitous, application-specific viruses began appearing, moving the attack away from the operating system to the applications being run. One infamous example is that of the MICROSOFT WORD CONCEPT virus [8]. Using the embedded scripting language, called WordBasic, the concept virus author created a program that would spread itself every time the recipient of an infected Word file opened that document. Released in 1995 to great effect, by 1997 anti-virus companies had developed successful defenses against this type of embedded attack.

One particular innovation, e-mail, was nearly transformed overnight from a relatively benign communications medium to an aggressively exploited virus transport mechanism. In 1999 the MELISSA virus, borrowing the idea of an embedded macro from CONCEPT, began spreading itself via an infected attachment over e-mail [9].

Once a recipient of a MELISSA-generated e-mail opened the attached Word document, the virus would e-mail itself to 50 addresses from the freshly infected system's Outlook address book. Estimates place the number of computer systems infected by MELISSA at 1 million and damage in excess of $80 million [10].

In general a virus is a program that requires the intervention of a user or an automated scripting process to execute on the vulnerable computer system.

1.2.2 Examination of a Worm

A worm, more so than a virus, is capable of spreading itself with great speed and without interaction with the users of a system. A successful worm is one that is capable of spreading itself at great speed, often by exploiting a known but as yet unpatched vulnerability in widely deployed software. The MELISSA virus is not strictly speaking a worm as it still required user intervention to open an attachment before the attack could be launched. In contrast, the SLAMMER worm, which automatically exploited a buffer overflow vulnerability in Microsoft SQL Server and the Microsoft SQL Server Desktop Engine, was observed as doubling in size (measured by number of infected host systems) every 8 seconds [11]. Globally, over 90% of the vulnerable systems were infected within 10 minutes of the worm's release. Worms that exhibit such super-spreading ability have been classified as *Warhol* worms [12]. While comparatively few hosts (75,000) were compromised out of all systems connected to the Internet, the impact was none-the-less severe, including disruption of ATM service, delayed flights and degraded ability to use network resources.

Worms pose a significant threat to the global network in its current form. Without automated and coordinated automatically-responding detection and mitigation mechanisms, the possibility for even greater disruptions and damage exist [13].

1.2.3 Honeypots and Honeywalls

A honeypot is a software construct whose value lies in being probed, attacked and compromised [14]. It is both a research tool that provides an early warning system for new and as yet unclassified attacks and a research tool that allows a network

administrator to discover computer systems that may be initiating unwanted contact with their network resources. Owing to this, a honeypot is, by itself, incapable of preventing traffic to real network resources and furthermore can only report on traffic that it sees. This thesis investigates how the idea of a honeypot can be evolved to a point where it can be deployed as an overlay to an existing network infrastructure with the intent of not only identifying, but classifying and preventing successful attacks. This evolved technology, the honeywall, is capable of both monitoring and responding to threats on the network.

1.3 Overview of the Research

What is it that we are going to do?

In this dissertation I will survey the current state-of-the-art techniques used in worm detection and mitigation. Each of the technologies will be tested against one another under identical conditions in a test environment created specifically for this purpose. In addition to this, I will create an implementation of a honeywall that is capable of detecting and responding to a worm attack in a timely and non-signature based manner that will also be run in the test environment.

It is my intention to demonstrate that, by taking existing honeypot technology and using it to populate unused IP space on a network, a honeywall could use these honeypots as sensors to detect and respond unauthorized traffic.

Who are we protecting?

The technologies reviewed in this dissertation are in most cases capable of scaling their deployments from a small network of a few single systems, to large enterprises that contain hundreds or thousands of computers.

While the potential exists for a honeywall to use a single honeypot as a sensor the true target site for deployment would be a company or university of medium to large size that has larger amounts of unpopulated public IP space. Having said this, the honeywall presented is scalable to the environment in which it is deployed.

Who are we protecting against?

1. Unsophisticated attackers. These individuals utilize scripted attack tools widely available on the Internet. While this type of person is not likely to cause significant amounts of damage in a well run network, they do have the potential to greatly disrupt systems that are not patched/maintained by a professional.

2. Worms/viruses. An increasing threat to network stability has been identified in the form of automated worms and viruses set to discover and exploit vulnerable hosts. Once a vulnerable site is discovered, a small payload is delivered. The payload forces the infected host to start probing and exploiting other machines, both on local and wide area networks. Regardless of the destination system vulnerability, these payloads still pose a threat to network stability and general network performance.

Who would we like to protect against?

The sophisticated attacker. Professionals of the hacking world, they are the few that have the knowledge and skill to discover and utilize new vulnerabilities. Once discovered, they may release a scripted tool to allow others to easily exploit a vulnerability. However, as a result of their expertise, they will also be the most likely to detect a honeywall in place.

Hypotheses:

1. By populating empty IP space with virtualized "systems" acting as ultra low interaction honeypots we can turn this otherwise unused area of a network into a powerful tool for detection of rapidly spreading worms.

2. Furthermore, by doing so we can cause the attacker to waste resources on what amounts to non-existent network assets.

3. A threat, once identified, can be responded to automatically by the system thereby reducing the total number of compromised computer systems on a network.

Experiments:

Create a rudimentary honeywall appliance that can be deployed on a given test environment which will be able to respond appropriately to known and unknown threats. Once created, use known scripted toolsets to test. Observe if traffic can be detected and stopped before total network compromise occurs. Compare these results to those observed from other technologies in the same environment.

1.4 Contributions

I create and test a reactive network intrusion prevention system capable of identifying a threat of which it has no specific knowledge: a honeywall. A honeywall combines the inherent advantages of a honeypot with those of a reactive firewall in such a way that when it is overlaid with an existing unprotected computer network, real network assets could be protected by those fabricated by the honeywall. This honeywall technology is compared against the results of several other accepted threat detection techniques and is shown to be a viable solution to the rapidly spreading zero-day worm problem space.

1.5 Overview of Thesis

This thesis is organized into two primary areas. The first areas comprised of Chapters 1 through 3 provide a basis for discussion of the technique developed in this dissertation. The second area, which includes Chapters 4 through 6, show the results of the honeywall method for detection and isolation of worms on a network, in addition to a discussion of the significance of these findings.

Highlights of this dissertation include:

- A high level overview of the critical need for accurate and timely detection and isolation of worms on the network.
- A review of existing worm detection and isolation techniques.
- A novel implementation of a honeywall, which demonstrates the ability to protect a vulnerable network from rapidly spreading worms.

Chapter 2 provides an overview of the necessary background for the concepts used in this dissertation. Chapter 3 is a review of the current state of the art in network security techniques, providing details on how these techniques are used today. Also included in Chapter 3 is a discussion of related works and the evolution of the honeywall mechanism created for this dissertation. Chapter 4 details the implementation of the honeywall, and provides comparison results for the honeywall mechanism against two other technologies used in worm detection and mitigation. Chapter 5 further discusses the results of the findings in this dissertation. Future directions for further refinement of the honeywall technique and the conclusion are found in Chapter 6.

Chapter 2

Background

An understanding of the nature and types of attacks seen on the network is established first. Following the classification of attacks, a profile of an attacker is presented to provide a more substantial example of network intrusions. A discussion of firewalls, intrusion detection, and honeypots is provided to complete the background information necessary to understand the topics discussed in this thesis.

2.1 Network Activity

Before we can attempt to detect attacks on the network, we must first specify what types of traffic and in what quantities are allowable. Specifically, through the exclusion of certain types of activities and patterns we will be implicitly creating a listing of anomalies that may be used as trip-wires for potentially undesirable network activities. The allowable network traffic profile can be translated into a policy, which will be further discussed in Section 2.2.

Lyle posits that most attacks fall within one of three main categories: attacks on integrity, attacks on confidentiality or attacks on availability [15]. The act of maintaining the integrity of a network is the act of preventing authorized users of the system from making changes beyond their authority, and to prevent unauthorized persons from making changes at all. If the integrity of a system can't be maintained, then the attacks on confidentiality and availability are much more likely to succeed.

Any data stored within a system whose access has been restricted to a set of users can be thought of as confidential. Within a computer network, multiple individuals

are performing roles where data access should remain restricted. For example, a researcher who is developing a new vaccine may want to limit access to her research while the study is being conducted and should be able to ensure that only she can access that data.

The ability of a system to remain accessible to authorized users is known as the system's availability or uptime. In the context of an e-mail delivery system, a system that claims 99.99 percent uptime will be unavailable to authorized users no more than 52.56 minutes in one calendar year. These claims are usually made without dispensation towards abnormal network activity, and as such can be adversely affected by the abnormally and artificially high loads generated by a denial of service attack.

2.2 Types of Attacks

Motivation for attacks are as varied and numerous as the potential attackers in the world, and will not be covered within the scope of this dissertation. The desired result of an attacker is to compromise one or more of the above listed principles of security. To accomplish their desires, the attacker must exploit weaknesses within the system they wish to compromise. On a given system any service, protocol or connected system can be viewed as a potential entry point for the attacker. The analogy of a chain is often used when describing the security of a system, in that the weakest member of the system will undoubtedly be the first to fail when tested.

At the start of an attack, all the potential attacker may have to go on is the IP address of the machine they want to compromise. While it is conceivable that they could run toolkits that try hundreds or thousands of known vulnerabilities against this IP address, the more than casual attacker will attempt to gather information about their target before launching an attack. Tools such as *nmap* [16] and *xprobe* [17] are widely accepted as being effective at probing a system remotely to profile the soon to be victimized system. Additionally, nmap and tools such as *p0f* [18] can provide operating system fingerprinting which can be very useful to the potential attacker, as it further refines the types and number of attacks that could be

launched. Once a view of the available ports, services, and operating system has been discovered, the attacker can begin to select from the variety of different attack mechanisms at their disposal.

2.2.1 Denial of Service Attacks

When an attacker wants to disrupt the normal operation of their target, but does not want to gain access to the system itself, they may choose to launch a denial of service attack. As the name suggests, a denial of service attack occurs when an individual attempts to overload the target computer's available resources which results in, at best, a drop in the quality of service that the system provides and at worst will cause the system to crash outright.

A denial of service does not necessarily require the attacker to have a large amount of bandwidth at their disposal. In a Distributed Denial of Service (DDoS) attack multiple compromised machines are used in concert with one another to simultaneously attack a target system. When the MyDoom.O computer worm was released it quickly spread itself to vulnerable computers running Windows [19], causing infected systems to target Microsoft.com. Once a system was infected, the worm sends itself to all e-mail addresses stored in any of the address books contained on the compromised host. Additionally, this variant of the worm was designed to query four major search engines to discover further e-mail addresses that it could propagate to. While the purpose of this worm was to create a distributed denial of server attack against Microsoft.com, however the consequences were far more reaching as the large amount of traffic generated by the DDoS affected several Internet Service Provider's ability to maintain their quality of service to their clients.

A distributed denial of service attack can affect more than the targeted system. As the number of infected machines grew, two of the search engines used by the MyDoom.O worm became overloaded with queries and eventually failed to respond to valid search requests. Additionally, each of the infected systems were forced to use some of their available network and memory resources while they participated in the attack.

By automating the attack through the use of self-spreading worms, the attacker is saved the time and resources of identifying and exploiting machines on a one-by-one basis [20]. The worm does not discriminate what type of host it sends itself to, be it a Windows system, a Linux system, a home computer or a financial institution's trading computer.

Denial of service attacks may not solely rely on overwhelming the bandwidth of the target service. An effective method to disrupt, for example, a VPN device may be to send it false Datagram Too Big messages [21]. If the attacker set the Path Maximum Transmission Unit size to be greater than it should be, and subsequently sends enough of these malformed packets, several datagrams will be lost, thereby disrupting the flow of valid network traffic. Another form of bandwidth independent denial of service is a SYN-flood which relies on sending a large number of extremely small TCP connection requests, thereby overwhelming the target computer's ability to process them [22].

2.2.2 Exploits

An exploit can be classified as the process in which an attacker goes about taking advantage of a weakness in their target's defense. An exploit that has been seen on computer systems outside of a test lab is said to have been found "in the wild."

Once a vulnerability has been discovered and successfully used to affect the target system, a common practice is for an attacker to identify this vulnerability to the Internet community. In some cases, the attacker is a concerned individual who has discovered a vulnerability in a product and desires the organization responsible for said product to issue a repair to close the vulnerability before it can be discovered and used by other less ethical persons. Several of the vulnerabilities discovered in various Microsoft products have been discovered and reported in this manner [23].

In other cases, however, the attacker will create a tool that allows less skilled or knowledgeable persons to exploit this vulnerability in a repeatable and controllable fashion. Once a tool has been released, the vulnerable software manufacturers must race to create a patch to close the vulnerability before it is widely exploited.

As will be discussed in further detail in Section 2.4 Intrusion Detection Systems (IDS), it is possible to create a signature of a known exploit that will allow an IDS to recognize previously discovered and published vulnerabilities as they occur on the network.

Vulnerabilities may lie undiscovered for years. When a vulnerability is found the discoverer must decide whether to release details of the vulnerability to other concerned individuals, or to attempt to enter a dialogue with the vendor responsible for the product. Some vendors argue that the likelihood of a vulnerability being exploited before a patch can be created is greatly reduced by not releasing the vulnerability to the community in the first place [24] [25]. However, those in the security community are often quick to point out that some vendors are slower to create patches for vulnerabilities told to them through "responsible disclosure" methods. In a mildly philosophic way the question is posed: If no one is aware of the vulnerability, does it really exist?

The most dangerous type of exploit is one that has not been published but has been discovered. No IDS signatures exist to mitigate the risk, no patches are being worked on by the responsible vendor, and the likelihood of the discoverer/attacker being able to freely use this exploit to their own ends with relative impunity is very high.

2.2.3 Buffer Overflows

Your home may have a wall socket that contains two identical electrical outlets. These outlets probably have been built to comply with your country's particular outlet requirements. What may be standard in one country may not physically fit in another country's outlets. The specification of the plug type can be likened to a data type. The placement of two outlets in a wall socket can be likened to having a two-element buffer with the same data type. We define a buffer to be a contiguous allocated piece of computer memory that is used to store one or more instances of an identical data type [26]. C programmers may use the words buffer and array interchangeably as in C the concepts are interchangeable.

In C, an array can be declared as static, where the size is determined at compile time, or dynamic, where the size is determined at runtime. At runtime a dynamic array, in the form of a variable, will be placed on the stack. For the purposes of discussion we will assume that when multiple dynamic variables are used, they will be layered on top of previously declared ones.

Much like a fluid-containing vessel, if too much data is poured into a buffer its capacity to store data will be exceeded and some of the excess will necessarily flow over the top. When a buffer has been filled with too much information it is said to have experienced overflow. The difference between a physical vessel and the computer's stack is that when the buffer is over filled, the data does not disappear down the sides of the buffer onto the ground. Instead, the data continues to fill the stack where the buffer itself is stored. In some cases it is possible for the data being written into an over-filled buffer to be written over the data in another buffer adjacent on the stack.

While buffer overflows can cause system instability through the unintentional destruction of valid instructions and data on the stack, malicious persons can use this same side effect to force a computer into doing what they want, often without the user's knowledge. Once an attacker has discovered a buffer overflow they may be able to use it to inject their code into the running code on the target system.

Several buffer overflows have been discovered in Microsoft products with varying levels of consequence [27] [28] [23]. One such vulnerability resulted in the creation of over seven different viruses and exploits in a four month period [29].

2.3 Firewalls

When constructing a building, especially when there are other adjacent buildings, one consideration that the designers must take into account is the potential for fire to spread between areas. If a fire was to break out in any given unit, there should be physical walls in place that impede the spread of the fire to connecting units. The term firewall in relation to computing refers to a barrier that in some way isolates one or more computers from others at the network level. Unlike a physical wall

that prevents both the spread of fires and impedes the movement of objects and people, a firewall can be configured to allow good traffic through and stop unwanted or unexpected traffic before it reaches its intended destination.

A firewall can be constructed in software, or as a standalone device that is placed physically on the network. A personal firewall is typically a piece of software installed on a desktop machine that allows the user to decide which applications are allowed to converse with the network. Personal firewalls can be deployed in a corporate environment where a higher level of security is needed, but are more typically installed on networks where a central network level firewall is not in place or possible. For example, a home user with a high speed Internet connection might install a firewall that only allows connections initiated from their computer to interact with their system. By doing this, all network traffic that does not originate or is requested by their desktop will be stopped before it can be processed by the traffic's intended destination. To further use this example, with the rise in automated worms and viruses that explore and infect the network automatically, a personal firewall may prevent an otherwise vulnerable system from becoming infected.

While personal firewalls give their users a great deal of flexibility to customize their installation, their disadvantages manifest themselves particularly in widespread or large networks that are centrally administered. For example, a department within a University may have 200 computer systems and one or more system administrators. If each of these two hundred systems have a personal firewall installed on them, the risk that a non-expert user may create a rule that prevents a valid application from conversing with the network is very high. Also, it is often possible for the users of desktop computers with firewalls installed to be granted privileges that allow them to modify their firewall configuration, which is another vector for a firewall related failure. Two solutions exist to the problem of misconfigured personal firewalls in the corporate environment. One is for the system administrators to be responsible for the proper operation of each and every one, a load that, even with user training, may prevent the administrators from accomplishing their other tasks. The other is to create a centralized firewall that encompasses the entire department's network.

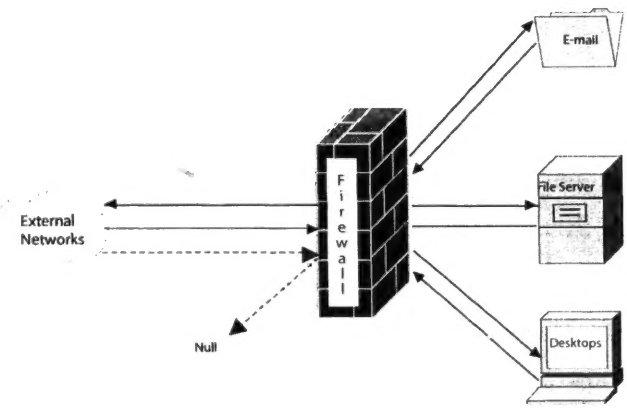

Figure 2.1: Example firewall placement

In a centralized configuration the firewall typically resides at the point where the network feed for the department is received. Figure 2.1 shows how a network level firewall might be implemented. Once the firewall is in place, the administrator must create rules that are sufficient to allow valid transactions to take place on their network while still providing some layer of protection from the outside world. While the mechanisms to create and modify these rules differ with the implementation, the methods are very similar.

A *whitelist* contains a listing of systems and/or ports that are allowed to pass through the firewall without further scrutiny. If the administrator knew that all traffic from the 129.128.10.X network was valid and should be trusted, this IP block could be added into the whitelist. Whitelisting mechanisms can be effective at preventing users from experiencing unexpected behaviour from their applications when said applications use an undocumented or obscure port to communicate. While the whitelist encompasses the known good hosts, a blacklist enumerates the known bad ones. Use of a blacklist can be effective at blocking systems that have historically shown malicious or unwanted activity originating from them. A blacklisting mechanism ensures that no network activity from the blacklisted hosts will be able to reach the production network that the firewall protects.

As a general starting point, a firewall configuration may allow all traffic outbound and deny all inbound traffic. This is facilitated by the statefulness of the firewall.

A stateful firewall has the ability to establish a transaction history of connections. If a machine behind the firewall (129.128.10.3) sends a request to an IP outside of the firewall (66.35.250.150), the firewall will record this transaction request. When 66.35.250.150 responds to 129.128.10.3's request, the information will be passed through.

To further improve the resolution of the filtering that a firewall can provide, mechanisms exist to block specific types of traffic on arbitrary ports. On a given network, the administrator may be able to determine that no UDP traffic should ever be sent and received on ports 1024-2048, and thus they could create a rule that would block that but still allow TCP traffic through.

Firewalls do not provide a perfect solution to the issues surrounding abuse and network attacks. Stemming from a misunderstanding of the scope of the technology, it is not uncommon for systems behind a firewall to remain unpatched or unsecured, a failing that can still be exploited. A firewall can't stop traffic that it has been configured to pass through. As discussed earlier, one of the predominant methods for worms to spread themselves is via e-mail, and e-mail is often considered a critical service on the network. Even though the firewall may be protecting the network from port scanning, buffer overflows and other malicious traffic, it will not be able to protect against other types of attacks. Furthermore, once a system behind the firewall has been infected, there is nothing in place to prevent that system from infecting other vulnerable systems also behind the firewall.

2.4 Intrusion Detection Systems

A system that attempts to identify and classify unauthorized activities, be they through use, misuse or abuse of the system, is defined as an Intrusion Detection System (IDS) [30]. An IDS may be configured to monitor a single system, but more often they are to monitor several. No one method of implementing an IDS has proven itself to be the best approach, a conclusion that is supported by the wide variety of solutions available. An IDS can be installed on a single system to monitor activities at a system level, be present on a system or device that monitors

the activity of a network, or may be a combination of the two approaches; that is to say a single system IDS with additional information provided from the network layer [31].

Of the network-level IDS systems present, two subclasses are available [32]. One captures data from the network and compares these snapshots or frames of data against a database of known attacks. This method of detection is known as pattern matching, signature analysis, or string matching. The effectiveness of signature analysis is largely dependent on the scope of the signatures that the incoming data stream is being compared against. If a data stream can't be matched to a pattern present in the database of signatures, no warning will be thrown. One caveat regarding a string matching IDS should be noted. Substring matching may also be a method used when analyzing a data stream for potential threats. As such, the possibility of a false positive increases as the substring size shrinks. A false positive is defined as an event that is detected as being malicious when it is not. Pattern matching can be done at a relatively lower cost to system resources than other techniques, and therefore a signature analysis type IDS can be scaled to handle large bandwidth networks with relative ease.

Alternately, a pseudo-intelligent IDS will still capture the raw data stream from the network, but instead of matching against known suspicious patterns, it attempts to emulate the destination's host and application based on the traffic seen. This emulation can provide a reduction in the number of false positives, and also may be more capable of handling attacks which would confound string matching IDS. However, the emulation is very resource intensive which prevents this type of IDS from easily scaling to meet high bandwidth deployments.

While a great deal of effort has been spent improving the field of IDS, by the very definition of an IDS a severe shortcoming is apparent: an IDS will at best correctly identify data as being a threat and will do nothing to actually stop a potential threat, and at worst will trigger too many false positives for the operator to identify the true threats [33] [34] [35] [36] [37] [38] [39]. The classification of a threat is important. However the damage may already be done.

2.5 Honeypots

A honeypot is a set of services presented to a network, often by one or more systems, that are not considered to be part of the production environment. These services are designed to mimic or simulate programs or even entire systems that a potential intruder may find interesting. Once an intruder is enticed, a honeypot may then interact with the potential intruder in such a way as to convince them that the honeypot is legitimately running the service that is being queried. While the depth of interaction will differ based on the application, a highly interactive honeypot's success may be gauged by the time that the potential attacker spends interacting with the simulation before realizing that they have been duped. By instrumenting a honeypot in such a way as to log these interactions, it can be used to monitor the attacker's activities for research purposes or for the purposes of intruder detection. Specifically, because no legitimate network traffic should ever reach these systems, all traffic reaching the honeypot can be viewed with suspicion [14].

The concept of a honeypot can be implemented using many different types of operating systems and hardware platforms. For example, the *honeyd* software currently supports FreeBSD, OpenBSD, Solaris and GNU/Linux operating systems [40]. However honeypots are not necessarily a product or application. Depending on the level of interaction your particular honeypot needs to have, it is possible to implement a basic honeypot using tools already existing on common operating systems.

2.5.1 Low Interaction

A low interaction honeypot is an incomplete or partial simulation of a set of services or operating systems that, upon cursory inspection, is sufficiently convincing to entice an attacker to proceed further [41]. As the attacker proceeds to communicate with the mimicked systems, their interactions will be limited by the depth of the simulation itself. For instance, an emulated service might be created to appear as if a given machine is running an open mail relay server, that when exploited would allow an attacker to pass mail requests anonymously through the "vulnerable" service.

This emulated process may only respond to telnet to port 25 by returning the start of a typical SMTP session, or may support further interaction via common SMTP commands. Lower interaction services typically reduce the chance of an attacker actually compromising the service itself as the service emulation is not the actual service that the attacker will think they are exploiting.

2.5.2 High Interaction

High interaction honeypots are well suited to capture hostile activity targeted at a network as they are usually composed of one or more real computer systems that are deployed in a known vulnerable configuration. Instead of closely modelling a system that an attacker can interact with, the exact system is provided. A researcher interested in detecting a new rootkit for Windows 2000 Server running IIS 6.0 would create the exact environment that they wanted to monitor. Merely creating an environment is not sufficient however, as this environment must be set up in such a way as to provide as extensive a forensic trail as possible prior to being deployed. Without instilling the research environment with a level of logging, at least equivalent to the perceived risk of deploying a high interaction honeypot, the full benefits will not be realized.

2.5.3 Honeypots in Practice

The benefits of the lowered interaction model, specifically ease of deployment and reduced risk of real system compromise, are at least partially countered by the limitations of low interaction. A simulation, no matter how complete, will eventually reveal itself to a sufficiently determined and knowledgeable attacker either through missed features, or through interaction that slightly differs from that of a known system. This limits the depth of logging that a low interaction honeypot will be able to provide. In practice a low interaction honeypot is better suited to the task of identifying potentially hostile IP addresses rather than providing an in-depth view of the attack against a system.

Through the deployment of a fully configured production environment, the re-

searcher does not restrict the possible avenues for compromise or detection by the attacker. For example, the researcher interested in how her IIS 6.0 computer could be compromised may see an attacker initially ignore IIS 6.0 entirely and proceed to compromise the system via another service running on the machine. This freedom coupled with high levels of logging are the true strengths of a high interaction honeypot and make them ideally suited to research of threats.

However, with the increased possibility of compromise comes the realization that a high interaction honeypot, once compromised, might be used against non-honeypot systems. For the purposes of our research we will not implement a high interaction honeypot, as our solution views every system on the protected network as a potential high interaction honeypot with poor logging.

2.5.4 Sniffing

It is three in the morning somewhere when a call is made from the personal cellphone of one of the world's most notorious diamond thieves. Instantly the investigators sitting in wiretap room are at full alertness, listening to and recording every piece of information exchanged during the call, ready to call in the troops if their suspect says something damning. In the world of computers, the ability to perform a function similar to the wiretap in the example is given by a device called a packet sniffer. This device can be inserted transparently in the network in one of two ways. Inserting the sniffer inline in the network provides the opportunity to both monitor and to modify the flow of data as it passes through the device. Inline sniffers run an increased risk of detection/corruption as all data must pass through them before it can continue on to its destination. Alternately, a sniffer can be installed on a span or tap port within the network. A span port receives a copy of the data flowing within the switch, which in turn allows the sniffer to see all of the traffic on the network without influencing the actual data flows. Direct detection of an span port sniffer is almost impossible from the end-user's perspective.

Much like the ability to tap a phone conversation, the ability to sniff packets can be used for both legitimate and questionable activities. By providing a view of

the actual packet traffic exchanged between computers on the network to a specific problem machine, the LAN administrator has the ability to reconstruct the unfragmented transactions, often essential when resolving connectivity issues. A packet sniffer so configured has the ability to monitor all traffic from the network to a specific machine in addition to all traffic leaving the specified machine.

When a packet sniffer is placed in promiscuous mode, it is capable of monitoring all traffic destined for the network interface it is running on, in addition to all other traffic on the subnet that the sniffer is a member of [42]. An attacker might use a packet sniffer to gather information such as plain-text passwords, logins, and e-mails, or to discover new IP and MAC addresses. In the case of an attacker using a packet sniffer to view a network's traffic, what may be completely legitimate traffic over authorized protocols can become a good source of material for the attacker to compromise resources on the network.

A packet sniffer in promiscuous mode can be very difficult to detect on a network, despite tools like *AntiSniff* from L0pht Heavy Industries [43]. A common misconception is that packet sniffers can only be used on unswitched networks. However by using techniques such as MAC spoofing [44], ARP spoofing [45], and impersonating the local network gateway [46] [47], or tools such as *fragrouter* [48], an attacker is still able to log network traffic. Tools such as the *dsniff* package exploits for many of these vulnerabilities into an easy to use utility [46].

2.5.5 Scanning

> To be certain to take what you attack, attack where the enemy cannot defend. – Sun Tzu

Though the Art of War was written more than two thousand years ago, the wisdom of General Sun Tzu is still relevant today [49]. The teachings were intended for the military elite of his time but have come to be applied in all aspects of life where an absolute victory is essential. When an attacker wishes to compromise a target, the wise attacker will attempt to determine his target's strengths and weaknesses. Once found the wise attacker will then exploit those weaknesses to

press their advantage against their targets. In the field of battle a general might send forth scouts to keep him informed of the enemy movements and strongholds. A digital attacker will also check the defenses of her target prior to launching a targeted attack. One such mechanism to reconnoiter a target is to perform a scan of their systems.

Before the Internet became as pervasive as it is today, computer systems used the telephone networks via devices called modems. A computer may reside in a physically impenetrable building deep inside a corporations' headquarters, but could still be potentially accessed from the outside world if it had a modem connected to it. When the telephone number this modem was attached to was called, the modem would answer with a carrier signal and thus try to establish a data connection between itself and the caller. The phone numbers for these systems were often unpublished outside of their specific user-base, as it was hoped that only those who needed to know about the system's existence would ever be able to access it. With the prospect of being able to directly connect into a major corporation's sensitive systems, attackers had the incentive to try locating these phone numbers.

Mass marketers long ago realized that if they dial enough phone numbers, eventually they will reach one that will get answered. Once answered the marketer would launch into their sales pitch hoping that the listener would buy whatever service or product the marketer was selling. Rather than going through the tedious process of manually dialing each of the target phone numbers, a device was created that would automatically dial a sequential range of numbers for the marketer.

Taking this automated ability to dial large ranges of numbers automatically to heart, tools such as *Toneloc*, so called war-dialers, were created that were capable of sweeping through thousands of phone numbers and recording those where a modem answered with a carrier signal [50]. Attackers quickly realized that by using these war-dialing programs they could effectively harvest a list of potential victims.

In a more modern context, attackers still need to identify potential victim computer systems and potential entry points into those systems. Connected via networks like the Internet, many of the computers are still vulnerable to brute-force techniques

that establish an unsolicited connection to them. These scans can contact tens or even hundreds of thousands of systems in a very short period of time by sending a large amount of packet traffic representing differing protocols. In the case of a tool such as *nmap*, probes can be sent and further details can be deduced about the target systems by the types of responses that are received via comparison to known signatures [51].

2.5.6 Fingerprinting

Since most security holes are version specific, the discovery of a target computer's operating system is of great value to a potential attacker. If, for example, the attacker probed their target and discovered that port 53 was open they might choose to launch an attack against the BIND daemon that is most likely running [16]. If the BIND daemon is vulnerable the attacker may only have one chance to successfully exploit it, as a failed attempt will most likely result in the daemon terminating itself [52]. By accurately identifying the operating system of the target host/machine, the attacker can better choose their exploit.

Chapter 3

State of the Art and Design

In Chapter 2 we introduce several technologies available to the LAN administrator. In this chapter we take a closer look at some of the state-of-the-art solutions available and how they apply to the problem space of worm detection and isolation. We also provide a discussion of honeywalls and how they are the next natural step in the evolution of reactive firewalls.

3.1 Firewalls

A firewall is defined as a device on the network through which network traffic flows. As network traffic passes through a firewall, the transactional information will be inspected and run against a ruleset. A network firewall is typically a separate piece of hardware which exists physically on the network, often located at an aggregation point. In the case of an office or departmental LAN, a firewall will commonly be placed between the outside world and the internal network as shown in Figure 2.1.

While the idea of a device that enables the good traffic to flow and prevents the bad from reaching the intended target is a simple one, there are several differing approaches on how to actually accomplish this goal. Firewalls appear in many guises, and in many locations across the network topology. In the following sections we examine and discuss the categories of firewalls, how they are deployed, and offer some insight into how they accomplish their tasks.

3.1.1 Border Firewalls

The assumption that attacks can only come from the outside is outdated and outmoded. The modern network is an ever changing entity where machines are added and dropped from the overall topology as individuals move around. For instance if a user has been off-site with their computer system, there is no guarantee that they have been able to maintain any level of security on that system. While a system, such as laptop, is used on a foreign network, be it a customer's site, a coffee shop, or even at home, it is subject to the security precautions made by those networks. This means that it is possible that a laptop which has been used off of the corporate LAN may return in a compromised state. When the user returns and plugs their machine into the office network, their system becomes a trusted member of the network with all of the privileges afforded it by that trust. In the case where the recently reconnected system is compromised, that leaves the internal network vulnerable to compromise and infection from an inside source.

Border firewalls can play an important role in a computer network by acting as a gatekeeper which permits or forbids traffic from passing through it between the external and internal networks. From a policy standpoint, border firewalls can be used to create a digital barrier through which depermitted traffic can't flow by blocking ports and IP ranges. However, the wholesale blocking of ports may be a contentious issue in some environments such as academic institutions where the users may expect unfettered access to the network. Several commercially developed solutions exist to perform the role of border firewall from such companies as Cisco, Sonicwall, 3Com and Radware. These solutions often are packaged as appliances which, while their implementations differ, are all capable of acting as a border firewall. The open source community has also developed a large number of software packages capable of being a border firewall. Each of the major open source operating systems have border firewall capabilities through packages like iptables, netfilter, ipfw, and pf. The testing performed in this dissertation was performed using pf as it was incorporated in the operating system used in the test environment; any

modern border firewall would suffice.

3.1.2 Host-Based Firewalls

A host-based firewall is deployed as a last and first-line defence against the network at large. As identified in Section 3.1.1, a border firewall can only protect the internal network from the external world but not protect the internal network against itself. By deploying firewalls on each of the member nodes of a network, each node becomes responsible and enabled to protect itself from every other node both on the internal and external networks. As seen in Figure 3.1, sometimes it is not possible to deploy a firewall on all of the devices on your network.

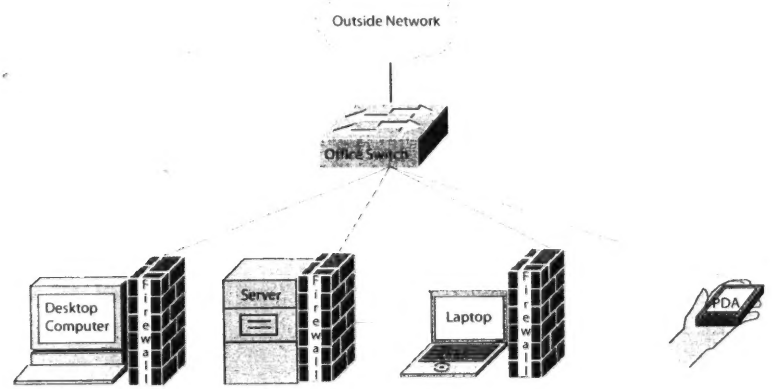

Figure 3.1: Host-based firewalls

An early form of firewalls still seen today is that of the stateless device. Traffic is permitted/denied on a per packet basis as those packets are matched against a ruleset. The term "state" refers to the firewall's ability to remember that a connection has been initiated by the host device. Stateless firewalls are uncommon today because they have extra rules to permit solicited responses to outbound requests. As a result these rules often create the possibility for an unauthorized inbound connection from the permitted ports.

Conversely, stateful firewalls are deployed in environments where the internal device or network is trusted by the LAN administrator to generate good traffic. If a system on the inside portion of a stateful firewall established an outbound connection

to a service over a port not explicitly permitted, then the firewall will remember this and when the response comes in from the outside asset its connection will be permitted. By monitoring the connection states of the internal protected network, the firewall can prevent unsolicited traffic from the outside network from reaching the internal private network.

Windows Firewalls

With the introduction of Windows XP Service Pack 2 (SP2), the Windows operating system began shipping with a rudimentary host-based firewall. The Windows Firewall is an example of a stateful low-interaction firewall. All outbound connections are allowed, and all inbound connections are disallowed unless they have been approved or "excepted" on a per application basis by the user. A startup policy is applied by the Windows Firewall as the computer loads allowing the computer to obtain IP and domain information and to establish itself on the network. The startup policy is transitory as it is removed once the computer loads the Windows Firewall service.

The Windows Firewall is low interaction as it will only ask the user to take action if unsolicited traffic is associated with an application running on the machine. The user is given the ability to deny the traffic, to allow the traffic (and implicitly create an entry in the exception table allowing this traffic in the future), or to temporarily ignore the traffic. If no association between traffic and application is made, the Windows Firewall will silently drop the traffic before it reaches the host machine. Also, Microsoft offers an option to not allow exceptions to the stateful rules. When this rule is activated the user will never be prompted to allow inbound traffic that may be unsolicited and all such traffic will be dropped silently.

While the Microsoft Windows Firewall is a good first step to protecting desktop computers running the XP operating system, it is far from a perfect solution. If a computer system becomes compromised any traffic generated by a worm process originating from the compromised host running the Windows Firewall will be allowed onto the network. As all outbound traffic is silently passed by the firewall the user

will not have any direct feedback from their host based firewall alerting them to the unusual activity.

Third-party Windows Firewalls

Applications such as ZoneAlarm and BlackIce Defender provide an increased degree of control over the traffic flow on a computer. They can bidirectionally apply policy on the installed host that can permit/disallow traffic both from outside networks as well as that traffic which originates from the host. Similar to the Microsoft Firewall these products use modal dialog boxes that prompt the user to correctly identify traffic/behaviour as events occur. The added benefit that can be derived from a host based firewall that is capable of applying policy on outbound connections is that the user can now control what applications are allowed to communicate with the outside world.

The ability to prompt the user and block outbound traffic can, when properly leveraged, allow for early detection of compromised host systems. In the situation where an executable has been installed onto a compromised host, a bidirectional firewall would usually prompt the user to see if this activity was expected and if it should be allowed. If the user identifies the malicious process as unexpected they can instruct the firewall to drop the traffic and prevent their compromised system from infecting other member nodes of their local network.

Some drawbacks exist with these systems. In their attempts to make deployment of their products easier, vendors often allow users to create policies that trust and permit all local network traffic. By permitting all local traffic, the firewall is no longer protecting local systems from one another, thus reducing their utility in the case of a major worm outbreak.

By enabling the user to make decisions on what applications are allowed to talk to the network, the software is implicitly assuming that the end user is knowledgeable enough to make the correct decision. For example, if a user is notified that"svchost.exe is attempting to communicate with 127.0.0.1", the program is expecting that the action the user takes is truly that which they want to take.

Managing large IP spaces can be very complicated. If the host-based firewalls are deployed in such a way as to allow individual users to manage the rules of the systems, they may complicate troubleshooting of connectivity issues. For example, if a user has chosen to create a list of hosts that are explicitly permitted upon installation of the software, they may not remember that the list exists later on when one of their coworkers not in the list attempts to connect to their system. This problem can be somewhat overcome through the creation of default policies that are distributed to all users of the host-based firewall mechanisms.

System resources are consumed by a host-based firewall. They are not optimized for high throughput attacks. Being software packages they consume host-system resources, both memory and CPU. In the case of a firewall that does some packet inspection that means that each packet that is passed to/from the host system will be inspected in software before it is placed back on the wire. As a result of this, high throughput network devices (such as fileservers), or systems that are being directly attacked with large amounts of traffic, may have their performance degraded.

3.1.3 Static Firewalls

A static firewall's rules, as the name implies, do not change. Specifically, any changes that are made to a static firewall's ruleset require the manual intervention from a privileged user. A static firewall is well suited to environments where rules can be easily codified and are likely to remain unchanged. Rule sets are chosen by the LAN administrator to suit their network environment. For example, if a LAN administrator knew that their servers should only ever see network traffic from the outside world on port 80 (web), then they could configure a ruleset on the firewall that would enable the firewall device to drop any traffic that was not destined for port 80 on the internal network.

Rules can be created on a per machine or per segment basis. It is not uncommon for a network administrator to conceptually separate the functions of systems within their network by function. For example, an administrator might group all of their servers together while creating another group for their workstations. By grouping

these machines together by function, rulesets can be created to apply to multiple machines with similar functionalities. Without any rules in place many firewall implementations will default to a "any/any all/all" rule which states that traffic from any source to any destination on all source and destinations ports will be permitted. When the "any/any all/all" rule is in place all network traffic will flow through the firewall undisturbed.

Therefore, it is considered a best practice to create rulesets from a default deny baseline, the idea being that no traffic will flow until the administrator has explicitly permitted ports and protocols. Static firewalls are limited in their effectiveness against threats such as worms. If a rule blocking the ports used by a worm does not exist prior to the release of a worm on the external network, the worm traffic will be allowed by the static firewall to pass onto the internal network unhampered. In the situation where a default deny-all traffic policy has been created, with exceptions added to permit traffic on an as-needed basis, the possibility of using non-standard ports to propagate is removed. In response, one of the mechanisms that is used by worm creators is to utilize ports that are commonly opened on firewalls to transport their payloads. For example, TCP ports 135, 137, 139 and 445 are used by several Windows applications and therefore typically have rules in place to permit traffic across those ports. By transporting the payload across a permitted port, the worm effectively bypasses the firewall, allowing it access computers on the internal network.

3.1.4 Reactive Firewalls

At a high level, a reactive firewall with no reaction mechanism is the same as a static firewall. Reactive firewalls are a superset of static firewalls in that they often are initially configured with a baseline set of unchanging rules. Over time, based on metrics specified by the administrator of the firewall, a reactive system may supplement these baseline rules with additional restrictions in response to events logged by the firewall. Reaction mechanisms range from manual intervention from the firewall's administrator, to automated processes that may be threshold or protocol based such as pf and netfilter [53], or Intrusion Detection Systems such as Snort

[54].

3.1.5 Intrusion Detection Systems

As noted in Section 2.4 Intrusion Detection Systems (IDS) have historically been deployed inline in a network making them capable of statefully inspecting and matching the traffic against known attack signatures as it flows through them. Many modern IDS systems are capable of reconstructing data flows to better process out-of-order attacks, as well as normally disrupted traffic. At its simplest, an IDS can be thought of as a string matching engine that compares all traffic on a network against a list of known bad strings that appear in various attack vectors, such as buffer overflows, worms, and e-mail viruses. Depending on the implementation, an IDS may be configured to passively log all signatures detected, or may be set to contact an administrator (usually once a set threshold of attacks has been reached).

Deployed inline on the network, a base install of *Snort* [54] is capable of recombining out-of-order traffic, and deep packet inspection looking for a wide variety of protocol anomalies, port scans, and host-based vulnerabilities. By comparing network traffic against a database of known attacks, Snort is capable of identifying threats as they arrive on the wire. This database of rules is currently maintained by the snort.org team and additional custom rules can be created by the end administrator to handle special cases/conditions within the environment it will be deployed in. Snort is currently billed as both an intrusion detection and an intrusion prevention system (IPS). When initially developed, Snort was capable of only reporting and alerting on signature matches. The ability to react to undesirable traffic based on the output of Snort was a feature initially added by other open source projects such as Hogwash [55], but has since been included and grown in the main Snort development branch. Over time Snort has evolved into what its creators claim to be "most widely deployed intrusion detection and prevention technology worldwide and has become the de facto standard for the industry" [54].

As with all active traffic suppressors there are risks and drawbacks to using Snort. Being such a high profile and widely deployed IDS/IPS, Snort attacks and scans are

created to bypass weaknesses within its architecture [56] [57]. Being largely signature based also has strengths and weaknesses. A well-written signature can have a very high detection rate while having a very low false-positive rate. False positives occur when a packet is flagged as containing a type of traffic that it does not. In situations where signature-based IDS/IPS devices are deployed in an active response mode, this can lead to valid traffic being dropped from the network. Because of the possibility to cause great damage to the valid traffic flows, the Snort team restricts their base signatures to well-tested ones with extremely high probabilities of successful detection. However, this often means that signatures do not exist for recently released threats. To fill in this perceived drawback, organizations such as Bleeding-Edge Snort [58] have formed, where community-developed signatures for a wide variety of applications aggregate in a freely downloadable and distributable form. The trade-off for using recently developed signatures is the decreased testing period and therefore the greater risk of valid traffic being flagged as bad.

3.1.6 pf

While not conventionally thought of as an IDS, the OpenBSD packet filter *pf* provides several features that can be applied to the problem-space typically associated with that of an IDS. Handley and Paxson [59] note that while a sufficiently robust and diligent IDS can address the issue of proper protocol inspection, it cannot correctly determine how the end system will process the packet stream. Pf allows the administrator to normalize the inbound traffic stream, which can prevent attacks deliberately fragmented by the attacker from slipping through the detection mechanisms by reconstructing the packet stream upstream of the analyzers.

Another feature that is exploited in pf is the ability to monitor the number of connections/states established by a system behind it. As a computer on the internal network attempts to contact a system on the outside network, the firewall creates an entry in the state table denoting that an outbound connection attempt was made. If a connection attempt returns from the outside system to the inside then traffic will be permitted. By taking a count of the number of states open in the state table on

a per IP basis it becomes possible to monitor how transactionally busy each device on the internal network is.

When deployed on the edge of a subnet, state table monitoring can be useful in identifying otherwise non-symptomatic compromised hosts. In a typical departmental LAN at the University of Alberta, it has been observed that most busy desktop systems have 25 entries, while busy web servers may have five to ten times that. This stands in stark contrast to worm infected hosts that may have as many as 5,000 to 10,000 entries. By identifying hosts with extremely high state counts, the LAN administrator is provided with a list of probable infected candidates to review and repair as necessary.

When pf is used at the border of the network, as is common in bridging and routing firewalls, it can only monitor outbound/inbound connections. This means that the firewall can't prevent infected hosts from reaching others on the local network. Also, implementing a threshold is only useful when the threshold permits normal traffic flow to occur unhampered. Until a threshold is met the infected host will be permitted to continue transmitting, even in routed mode. The risk of setting a threshold to high is that an infected computer system will be allowed to propagate unabated if the threshold is not exceeded. Conversely, if a threshold is set too low, legitimate traffic may be stopped even in the absence of an infection.

3.2 Honeypots and Honeynets

The Honeynet project [60] offers some off the shelf tools that members of their alliance can deploy and use to create networks of computers (real or virtual) that are deployed in such a way as to gather information about the types of traffic reaching the honeynet. Participants in the Honeynet project can aggregate the information gleaned from their honeynet deployments into a central repository, where information security researchers can review how a system was attacked and possibly compromised. A honeynet requires a large amount of resources allocated towards it to maintain as each of the honeypots are typically high interaction. Because the honeypots that are deployed in the honeynet are high interaction they may be sub-

ject to compromise, which can result in the honeynet falling under the control of an attacker. Technologies have been developed that can detect and disrupt honeynets [61] [62] which diminish their usefulness further.

Neil Provos' *honeyd* virtual honeypot is a daemonized service that creates virtual hosts capable of emulating a wide variety of services and operating systems. By using *honeyd* one computer system can appear as hundreds or even thousands of different computer systems on a network. By manipulating a configuration file, each of these virtual hosts can be set to emulate different services with different operating systems as needed. Fingerprints, as discussed in Section 2.2, are borrowed from nmap to provide the appearance that a virtual host is running a particular operating system. Provos has explored simulations using honeyd to identify and actively counter through forced patching worm infected hosts on a network [63]. Through these simulations he has shown that by actively patching known compromised systems that a widespread infection can be slowed down, or even stopped *if* patches can be applied in a timely fashion.

3.3 Honeywalls

A honeywall extends the notion of a honeypot by combining the capabilities of a low interaction honeypot with the abilities of a routing firewall. At its core, a honeypot is a research tool that can provide a more in-depth view on the timeline of an attack by interacting with the attacker in a manner that belies the true nature of the honeypot itself. Previously in Section 2.5 three types of honeypots were discussed, each with their own costs and benefits. Still, even in the highest interaction mode, honeypots are nothing more than passive participants on the network in that they can't actively modify the behaviour of systems around them.

When a zero-day worm strikes a network, conventional mechanisms of protecting the network fail. Border firewalls can't prevent client systems within a protected network from infecting one another. Host-based firewalls may be bypassed, rules may exist allowing local infection through trust relationships with other systems on the local network, and may be unable to handle the load presented by a rapidly

spreading worm. Signature-based detection mechanisms can't be relied upon in a zero-day worm outbreak as a delay exists between the initial infection run of a worm and the antivirus vendors releasing signatures to detect attack traffic.

By leveraging a low-interaction honeypot's ability to pose as a valid member of a local network with a reactive routing firewall's ability to act upon undesirable traffic, it was anticipated that a highly sensitive and accurate mechanism could be developed that would allow us to locate and isolate potentially infected members of a network prior to widespread infection.

To start, a honeywall is deployed on an existing network as a simple routing border firewall. This machine also becomes the aggregator for network traffic – all traffic on the network will pass through it before being returned to the local network wire. By passing all traffic through this routing firewall, a single point where traffic can be stopped is created. In practice this mechanism could be applied to a network router or switch, but for the purpose of the experiments a PC-based server was created to show proof of concept.

Once the honeywall has the ability to control the network, several honeypots are inserted into the unused IP addresses on the existing production network. In practice, as the honeywall system acts as a DHCP server for the local network, the knowledge of the topology can be used to sparsely distribute honeypot listeners on unpopulated nodes. For the trials performed in the experiments all unoccupied IP space was consumed by honeypot listeners. The honeywall uses the honeypots as listeners on the network to identify unexpected traffic flows and to pass the security event to the reactive firewall mechanism where the security events can be acted upon.

As is shown in Chapter 4, the honeywall mechanism has proven itself capable of quickly identifying and isolating rogue systems on a network in such a manner as to prevent widespread infection of even densely packed networks.

Chapter 4

Evaluation

Having established the current state-of-the-art worm detection and mitigation technologies in the previous chapters, it is now time to evaluate these mechanisms against one another. In this chapter the theories and some of the tools mentioned in the previous chapters are put into a test harness designed to evaluate the effectiveness of various worm detection and mitigation techniques. As stated in Chapter 1 the main hypothesis of the research is to determine if populating unused IP space with ultra-low interaction honeypots will make it possible to detect and isolate a rapidly spreading worm, thereby reducing the number of compromised computers on the network. To this end, a survey of various techniques for detecting and isolating infected hosts is performed in addition to the creation of and testing on a honeywall mechanism. The testing performed is not how each of these techniques perform against MS.BLAST, but against a generic worm process that exhibits characteristics common to fast spreading worms.

The single most important criteria in evaluating the effectiveness of a worm detection and mitigation technique is to simply count the number of vulnerable systems that have been compromised despite the defensive measures. Ideally a detection technique will be able to find and isolate an infected system before it has the opportunity to infect any other vulnerable systems on the network. A baseline for worm performance is established first by releasing the worm onto a network of fully vulnerable hosts. All solutions are evaluated against this baseline, both for the total number of vulnerable systems compromised and for time until all

infected systems are isolated. It is theorized that the quicker the detection/response mechanism is, the better it will be able to protect the network from infection. This chapter concludes with a summary of results and findings.

4.1 Test Harness

During the setup of the test environment it was anticipated that creating an automated tester would be desirable which could provide a method to rapidly test the following matrix:

Size of Network	Distribution	Infected IP	Netmask	Firewall	Virus	Rule Sharing
1x30	Sparse	First	24	Open	Blaster	Yes
1x51	Dense	Mid	23	Closed	Nimda	No
2x25		Last	22	Dynamic-Snort	Custom	
5x10			16	Dynamic-pf		
				Dynamic-Honeywall		

Table 4.1: Experimental setup variables table

If all possible outcomes were tested, 2,880 different trials would need to be run, as shown in Table 4.1. Further analysis of this trial set revealed that several reductions in the problem space can be performed.

Initially four different network sizes were thought to be necessary to be representative of common deployments in the real world. The 1x30 and 1x51 sizing would be representative of a small office or department and would help establish a baseline for comparison of the 2x25 trials. By creating a 2x25 sizing, it was expected that spread between two distinct networks could be shown. It was anticipated that a 5x10 sizing could demonstrate how a worm could move across multiple networks. However, it was realized that the number of Size trials could be reduced to that of a single boundary case – one where an "outside" machine attempts to connect to an "inside" machine. Specifically, while distribution and density of IPs on a network will vary from site to site, and the mechanism by which a worm will spread itself varies, a connection between a malicious and a target machine must be established. For example, if two networks of 25 machines each were created, one on Subnet A and the other on Subnet B, the boundary case of a machine from one subnet infecting a system on the other subnet can be reduced to that of an infected host connecting to

a vulnerable one. Thus for the testing trials were performed using one subnet with 51 computer systems unless otherwise noted.

Distribution of IP addresses to the physical computers was also considered, as it was theorized that by creating a separation between physical systems in the IP space the detection systems may be given an additional window of opportunity to detect and respond to a threat. A sparse distribution of IP space, which is to say allocating IPs that are spread apart from one another, may give additional advantage to detection mechanisms when a worm uses a linear scanning technique. A dense distribution, where all computers are assigned adjacent IPs, may offer some advantage to detection mechanisms when a worm uses a random scanning technique to propagate. However, both the linear and random scans of a local Class C network consisting of 255 IP addresses are accomplished at such great speed in a mass spreading worm as to negate any advantage of a sparse or dense IP distribution of a network. Therefore a sparse distribution has been chosen. Physical computers in the trials are allocated IP addresses roughly equal distance apart in the various network configurations.

Choice of initial infected host was also considered as a variable for testing. However, with a linearly scanning worm the initial infected host IP is irrelevant as the worm will choose a point in the IP space and start its scan from there. Likewise, in the case of a randomly scanning worm, the initial infected IP will not matter as the worm will randomly choose IP addresses within the local space to scan and infect. Therefore, for the trials performed the same IP address was chosen to be the initial infected host.

The total problem space to be tested was reduced based on the following considerations. An open firewall, that is to say a firewall with no rules or with "any/any all/all" rules, is equivalent to having no firewall at all and thus is covered by the open network tests. A static firewall that does not have pre-existing rules that block the undesirable traffic also is equivalent to having no firewall. A static firewall that has rules in place to block the specific test worm will not allow the traffic to propagate from an external source to the internal network but, as noted in Section 3.1.1 it will

not be able to prevent traffic from occurring within the protected network. From this analysis the total trial space was reduced from 2,880 possible combinations to 8 representative trials.

4.2 Performance Indicators

The metrics described in this section are used to assess the overall effectiveness of the worm detection techniques within the test environment. These performance indicators were chosen as they can graphically demonstrate the differences in how the varying techniques perform, as well as illustrate how quickly a rapidly spreading worm can affect a vulnerable environment.

4.2.1 Number of Systems Compromised

The total number-of-systems-compromised metric is interesting because it is the golden standard by which all products are judged – specifically the effectiveness of a protection mechanism can be determined by comparing the total number of vulnerable systems to the number of machines infected after the network has reached steady state post infection. The derived compromised/vulnerable ratio can be used to compare the effectiveness at controlling the spread of a virus within the test environment.

4.2.2 Time to Stable State

Stable state is defined as the state in which no further systems within the test environment are infected by other compromised systems on the local network. In the case of an unprotected network, measuring the time to achieve steady state provides a spread rate for the infection. Once established the baseline can be used to compare various protection mechanisms reaction times against one another. Stable state is reached either when all vulnerable systems in the test environment have been compromised by the virus, or when the spread rate of the virus within the test environment goes to zero.

As the response time of a protection mechanism decreases, it is thought that a

similar decrease in the time to reach stable state occurs. Response time is defined as the time between the protection system signaling that something is wrong to the time that the appropriate mechanism is put in place to mitigate that problem. However, as established later in the chapter, this was not the case.

4.3 Test Worm Configuration

Naturally one of the key components to the testing and verification of the various protection mechanisms is a consistent repeatable infection agent with known behaviours. The selection of a worm for the trials was heavily influenced by three factors: repeatability, ease of replication, and ease of detection.

To meet the repeatability criteria a worm must have a known behaviour that can be observed and measured by a common set of tools. For example, a worm that is known to attack a specific service or a specific set of ports would be valued more highly in the selection process than one which indiscriminately attacks a wide assortment of vulnerable services or has a random timer of days or weeks before spreading itself. It should be noted that while expert knowledge of the worm was necessary for observation and measurement, no specific information regarding the worms, unless otherwise specified, would be imparted to the solutions tested.

Ease of replication encompasses how easily the test environment could be reset to base state between infection runs. As a significant number of trials needed to be run during the testing, a reduction in time between runs was important. Initially the lab environment was configured to run Windows 2000 Professional in unpatched form. In this state the test systems were known to be vulnerable to a number of the mass spreading worms that were candidates for our testing. With some reduction in the installation image it was found that all 51 test systems could be reliably re-cloned every 15 minutes. However, it was necessary for each system to be manually verified during this re-clone process as two reboots were required for the system imaging to be complete.

To achieve the ease of detection goal, a reliable binary mechanism able to detect whether or not a system is infected is needed, preferably with a date/time stamp to

allow for reconstruction of the infection path at a later time if necessary. Additionally, worm infected systems need to start spreading soon after infecting a vulnerable computer. This ruled out worms with set propagation windows such as CODERED. Infection that could be remotely detected, or would signal an infection host, were given priority over those that would need to be manually verified on a per system basis. It was felt that installing an anti-virus tool on the vulnerable systems could interfere with the propagation of known worms, as most anti-virus packages will prevent the infection of a host if a signature has been added to their knowledge base. Similarly, a host-based anti-virus program without knowledge of the worm would be unable to detect the infection.

Initial tests were performed on a test environment running an unpatched version of the Windows 2000 operating system known to be vulnerable to a number of worms such as MS.BLAST, CODERED, NIMDA, etc. After examination of the characteristics of the known worms, MS.BLAST was selected for use in the experiments because of its fast-spreading characteristics and its well-documented behaviours. However, the results from this testing demonstrated several drawbacks with using a real worm in a test environment. In some cases systems were rendered inoperable by the worm. This complicated detection, as well as reducing the spread rate of the worm as inoperable systems would need to be identified and rebooted before they would start transmitting the worm. Also, remote detection was complicated by the fact that remotely scanning the systems gave inconsistent results, and a manual process would have to be run on each system to verify infection which may introduce errors in the data. If this process was scripted, then an increased load on the vulnerable systems would have skewed results by slowing the infection rate. Also, the spread time resolution between machines would have been skewed by the nature of the repeated batch process run time (in excess of 5 seconds).

It was felt that creating a custom worm that exhibited key spreading features of the MS.BLAST worm was the best method for meeting all three criteria. To this end, each of the clients systems were setup with a daemon service that was vulnerable to an exploit payload of identical size to that of MS.BLAST. When this

exploit payload was received, i.e. the daemon was compromised, the service would then start exhibiting infected behaviour. Table 4.2 below shows the key features of the MS.BLAST worm and the custom worm created for the testing.

Feature	Ours	MS.Blast
Target Port	5678	135, Listens 4444, UDP 69
Targets of Worm	Vulnerable host process	DCOM RPC (vulnerable dll)
Probability of Infecting on LAN machine	40%	40% **
Probability of Infecting off LAN machine	60%	60% **
Scanning Threads	20	20
Payload (bytes)	6197	6176
** Note that MS.Blast will send a Windows XP exploit 80% of the time and Windows 2000 20%		

Table 4.2: Feature comparison between MS.BLAST and the custom worm

The custom worm that was created for the testing exhibited a simplified feature set of the MS.BLAST worm. The custom worm operates in the following manner:

1. Once the 6,197 byte payload is received by the vulnerable listening process on port 5678, the process becomes compromised. For ease of signaling, the compromised host process prints the string "Eeeek! I feel violated..." once it has been infected by the payload. By sampling this string from the traffic stream, it is possible to locate and time when each vulnerable host process becomes compromised.

2. The compromised process forks 20 child processes. Each child is assigned a host within a Class B network with a 40 percent probability of being assigned an IP within the current Class B, and a 60 percent probability of the IP being outside the current Class B network. After each of the 20 forks a delay of 1/20th of a second occurs to avoid system resource exhaustion. As each thread terminates, it signals the parent process of its completion. After all 20 of the terminated threads have signaled the parent, 20 new threads are spawned by the parent and the scan continues.

It should be noted that while the custom worm created for testing exhibits some of the characteristics of the MS.BLAST worm, it needn't exhibit them all. For example, when MS.BLAST is propagating it will send the Windows-XP-specific

version of the exploit with an 80 percent probability, and a Windows 2000 exploit with a 20 percent probability. In a heterogeneous operating system environment, it is not guaranteed that all systems will become infected as there is no guarantee that the vulnerable system will receive the correct exploit. Conversely, the custom worm needs to only send one version of itself to a vulnerable machine. As all computers in the test lab are running a vulnerable service, they are all susceptible to compromise, and all infected hosts are guaranteed to infect a vulnerable system when they send the attack payload. The custom worm, therefore, is capable of spreading itself at a higher rate than the MS.Blast worm.

For the trials it is not necessary to recreate MS.Blast worm as it is only one of many types of worms that currently exist. Instead, certain characteristics such as spread pattern, scanning pattern, and payload size are mimicked.

4.4 Lab Configuration

4.4.1 Client Machine Hardware

Fifty one identical computers were used in the test environment, based on Dell Dimension GX240 desktop systems. Each contained a Pentium 4 1.8Ghz processor and 256MB of RAM. As discussed in Section 4.3 these computers were setup with a minimal OpenBSD 3.6 installation. Each client system was re-imaged after each test run to ensure that no residual effects from previous trials could affect subsequent testing. The client machine image contained a daemon which was configured to listen on a specified port. Upon receiving the exploit sequence, in this case a padded plain text payload, the daemon would become infected and start attempting to spread the virus, as described in Section 4.3.

Network connectivity was provided through a 100Mbit connection going back to one of two fiber connected Cisco 2924 switches.

4.4.2 Protection Hardware

The computer used as the installation base for the various detection techniques was a generic PC running a Pentium 4 1.6Ghz processor and 512MB of RAM.

This protection system had a full installation of OpenBSD 3.6 Stable. Network connectivity was provided by two 100Mbit network cards connected to the Cisco 2924 switch.

4.4.3 VPN tunnels

VPN tunnels configured to pass all network data were established from every client machine to the switch. These VPN tunnels could then be configured to allow the machines to talk freely to one another as they would normally be able to with a standard office network configuration, or could be restricted to speaking through the switch as the trial needed. By doing so the test environment could be physically configured once while still allowing the testing of many different network topologies.

4.5 Methodology

For all trials the sparse distribution of computers on the local network is the same unless otherwise noted. For each trial the same vulnerable system is initially infected with the worm.

4.6 Static Firewall

To establish a baseline for worm effectiveness, the first experiment is run on a test network where no reactive mechanism is in place to detect and respond to the worm and where no pre-existing rules are in place on the firewall. By doing so it is expected that the worm will infect all of the vulnerable systems in the test environment. Figure 4.1 shows how the network was configured for this experiment. In this test, each of the computers is permitted to speak with all others on the local network without hindrance from any packet filtering/blocking devices.

In Figure 4.2 within approximately 20 seconds, 10 of the 51 vulnerable host computers are infected by the worm. Within 40 seconds of initial infection 20 of the 51 vulnerable hosts have been infected. By approximately 70 seconds into the trial 50 of the vulnerable hosts have been infected with the worm, demonstrating this worm's ability to effectively and rapidly spread itself to all vulnerable host

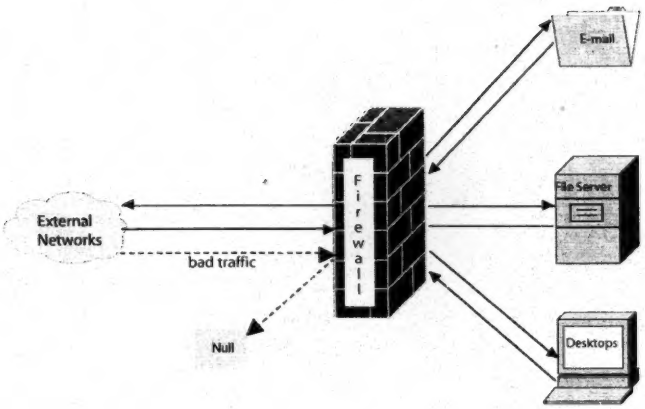

Figure 4.1: Static firewall deployment

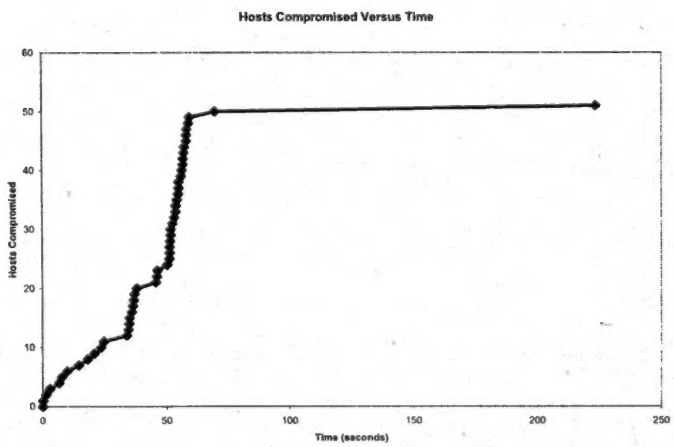

Figure 4.2: Static Firewall with no rule to block worm.

systems within a network. The amount of traffic generated by these 50 infected hosts was so great that all available resources on the Cisco switches were exhausted, reducing network performance to the point that the 51st and final vulnerable host was infected approximately 233 seconds into the trial. This underscores the severity of a worm attack by demonstrating how even a relatively small payload can be sent with such regularity as to disrupt the normal operation of a network, even if some or most of the hosts on a network are not susceptible to infection.

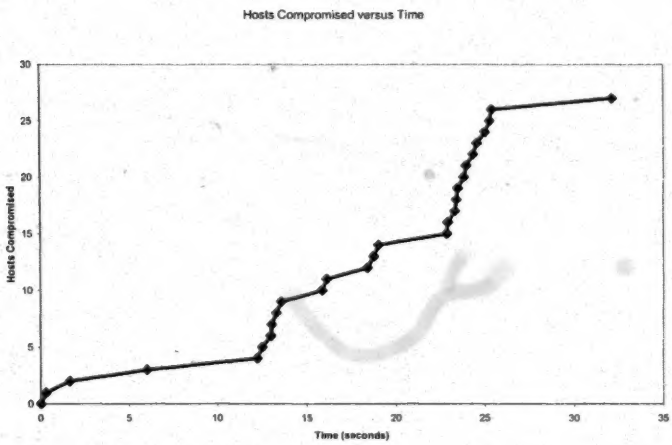

Figure 4.3: Two subnets of equal size were created with a static firewall between the two. A rule was put in place on the static firewall to block propagation of the worm between the two subnets.

To demonstrate that a static firewall could, given the correct circumstances, prevent infection from spreading between multiple subnets, trials were performed where two subnets were created, separated from one another by a static firewall. This firewall was configured to block the port which the worm uses to propagate. As seen in Figure 4.3 the static firewall with a pre-existing rule to block the worm traffic was able to prevent the worm from spreading from one subnet to the other, with the 26 machines on the initial infected network becoming infected. In the case where a network a has host-based firewall deployed, those with rules preventing

propagation of the worm would remain uninfected. However, it is important to note that the rules preventing propagation may interfere with normal operation of the computers on the network in cases where a worm propagates using commonly used ports as noted in Section 3.1.2. Therefore, in restricted circumstances a static firewall may be able to block a rapidly spreading worm, but only in the situation where the ports used by the worm to propagate have already been blocked prior to initial spread of the worm.

4.7 Reactive Firewall

For testing the Reactive firewall, three candidates were selected: a threshold based model using OpenBSD's pf, a signature-based model using Snort, and the Honeywall model described in Section 3.3.

4.7.1 IDS Using pf

Three thresholds were set for testing pf. The first experimental threshold was set at 50 connections established within a 4 second period. This number was chosen as it was roughly double the number of connections in a 4 second window that a typical desktop computer would see, and is therefore a reasonable upper bound for testing rapid worm propagation. By choosing a high threshold, it is expected that some, if not all, vulnerable computers on the network will become infected before pf is able to react and stop propagation.

The second experimental threshold was set at 8 connections in a 4 second period. This number was chosen as it is roughly half the lower bound for the number of expected connections from a desktop computer under normal usage. By choosing an extremely low threshold it is expected that fewer vulnerable computers will become infected than with higher thresholds.

A third experimental threshold was set at 28 connections in a 4 second period. This number was chosen as it is approximately half way between the thresholds set in the previous two experiments, and is representative of a threshold slightly higher than the normal number of states expected from a single uninfected desktop

computer.

The pf experimental setup is shown in Figure 4.4 below.

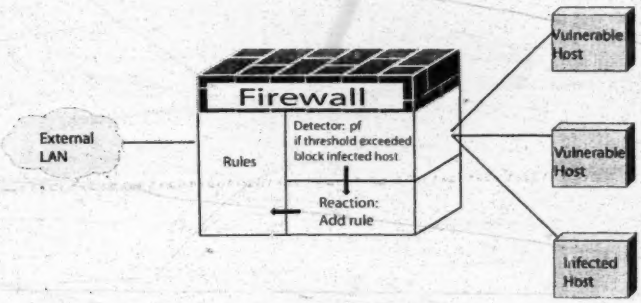

Figure 4.4: OpenBSD pf deployed as an IDS

Pf's state table monitoring mechanism was used as a tripwire to determine whether or not a machine had gone rogue. Specifically, if the number of states established by a member of the test network exceeded a specified number over a specified period of time it was deemed to have been compromised. Three levels of sensitivity were chosen. It is important to note that the X and Y scales vary on each of the result figures.

Level 1's threshold was 50 states within a 4 second period. (Loose enough for regular use.)

Result: On average 21 of the 51 machines were infected before stable state was reached. Figure 4.6 shows the best result obtained with this experimental setup (6 hosts in 263 seconds) while Figure 4.5 shows the poorest result (51 hosts compromised in 225 seconds).

Level 2's threshold was 8 states within a 4 second period. (Extremely restrictive)

Result: On average 2 of the 51 machines were infected before stable state was reached. Figure 4.7 shows the best result obtained with this experimental setup (1 host in 0.31 seconds) while Figure 4.8 shows the poorest result (4 hosts 57 seconds).

Level 3's threshold was 28 states within a 4 second period. (Medium restriction)

Result: On average 14 of the 51 machines were infected before stable state was reached. Figure 4.9 shows the best results of this experimental setup (2 hosts in 2.7 seconds) while Figure 4.10 shows the poorest result (38 hosts in 76 seconds).

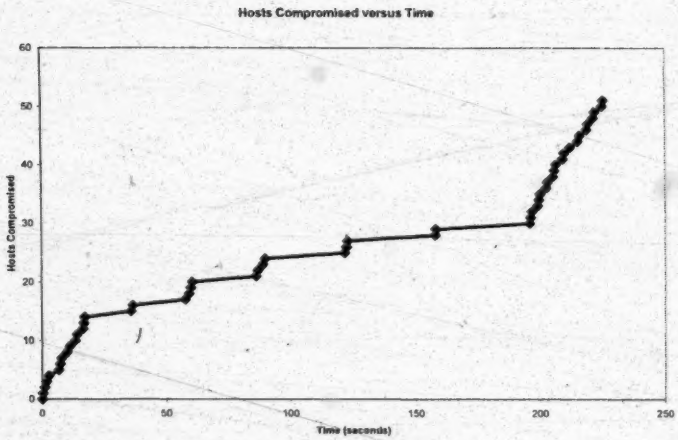

Figure 4.5: Worst case result from OpenBSD's pf with a threshold of 50 connections in 4 seconds per computer

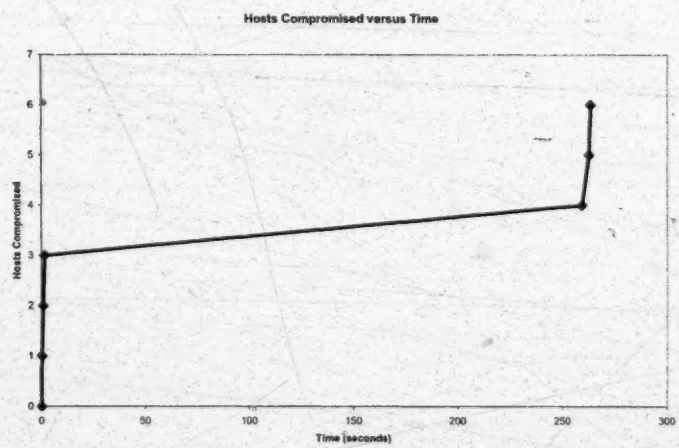

Figure 4.6: Best case result from OpenBSD's pf with a threshold of 50 connections in 4 seconds per computer

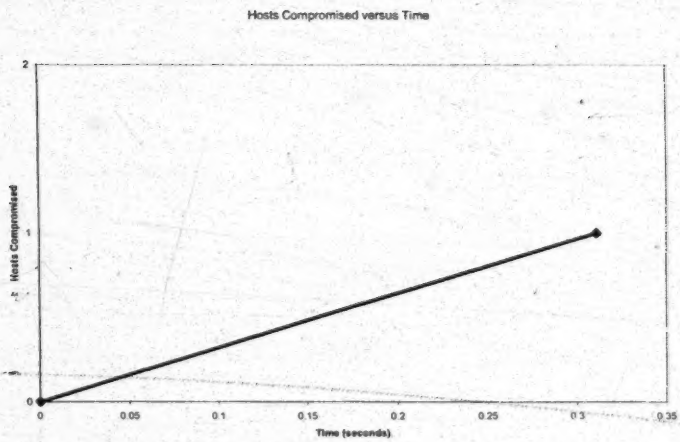

Figure 4.7: Best case result from pf testing with threshold of 8 connections in 4 seconds per computer

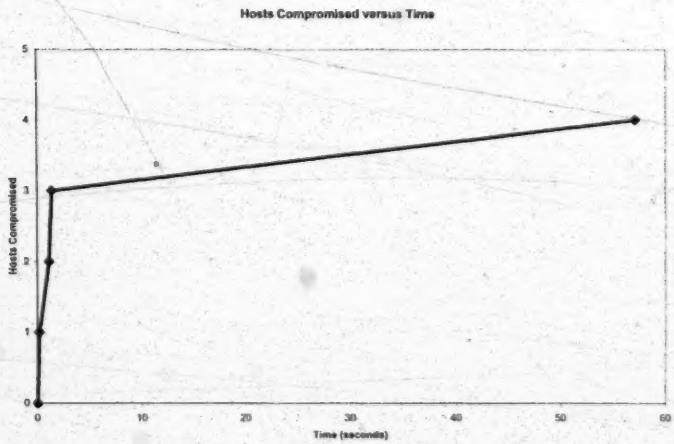

Figure 4.8: Worst case result from pf testing with threshold of 8 connections in 4 seconds per computer

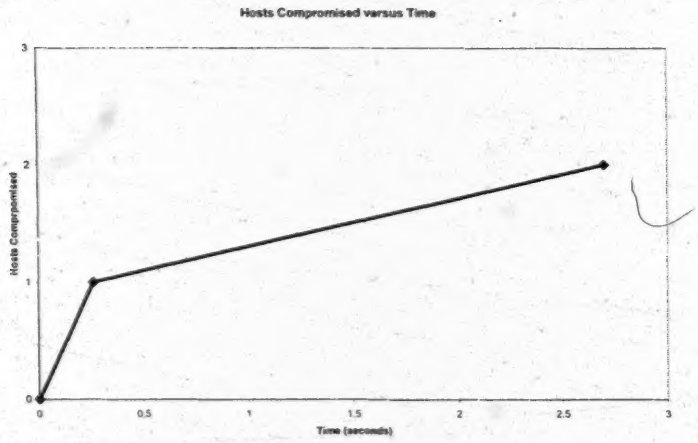

Figure 4.9: Best case result from pf testing with threshold of 28 connections in 4 seconds per computer

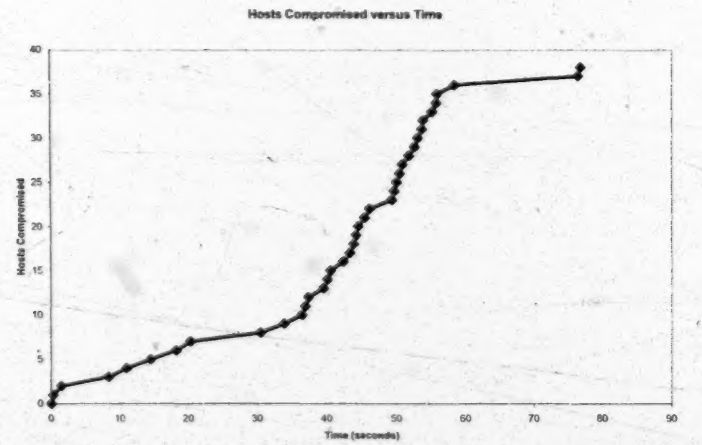

Figure 4.10: Worst case result from pf testing with threshold of 28 connections in 4 seconds per computer

From the results of these experiments we can conclude that a threshold based method for worm detection and mitigation can be effective under very special circumstances. Specifically, a threshold model may be able prevent a worm from infecting all vulnerable systems when that threshold has been set extremely low. From a practical standpoint however, using thresholds to stop worms will be impractical to deploy in the real world because, as shown in the testing performed, to stop a rapidly spreading worm effectively the threshold must be set below the average number of states that a normal desktop system would use. Such a low threshold would result in legitimate traffic and uninfected systems becoming blocked on the network.

An additional drawback of the threshold model is that it will be incapable of stopping a slow spreading worm. In the experiments performed here, the worm attempts to affect as many systems as it can in as short a time as possible, resulting in a large number of states being used by each infected client machine. Another type of worm that, for example, only attempts to connect to one system at a time would most likely go unnoticed by a threshold based detection system.

4.7.2 IDS Using Snort

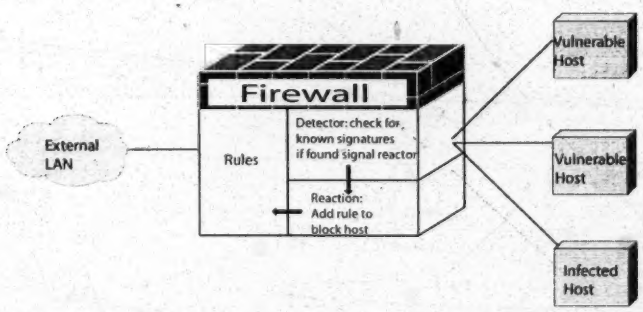

Figure 4.11: Snort deployed as an IDS

In the case where no predefined rule exists for the attack in Snort's database the test environment will perform similarly to that of an unprotected network. Therefore a custom rule was created to allow Snort to detect the payload delivered by the test virus. The more precise a rule's specification, the less likely you will receive a false

positive from the Snort IDS.

Snort was setup to send a message to the switch when data on the network was found to match a signature in the database. The reactive firewall acts on the message from Snort by blocking the network access to the machine of origin. Snort was configured to only view traffic inbound to the network, meaning traffic originating from the network going off network would not be detected.

Trial set 1 was performed with only the custom rule loaded in the Snort database. Figure 4.12 shows the poorest result of 3 compromised hosts in 1.74 seconds, while Figure 4.13 shows the best result of 2 compromised hosts in 0.387 seconds.

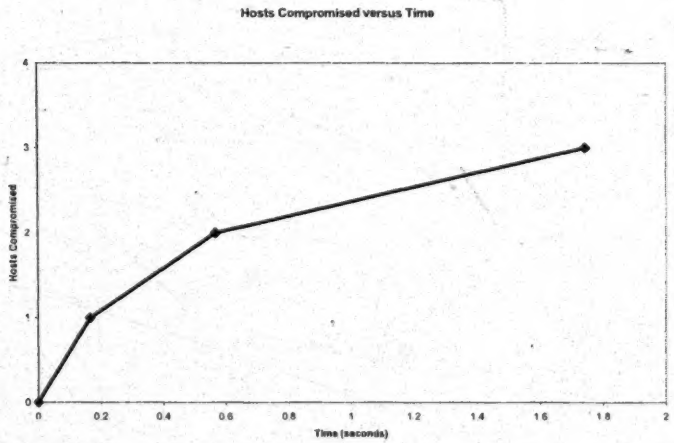

Figure 4.12: Worst case result from snort testing with only the custom signature loaded

Trial set 2 was performed with the entire base ruleset included with Snort installed in addition to the custom rule. Figure 4.14 shows the poorest result of this experimental setup (3 hosts in 36.62 seconds), while Figure 4.15 shows the best result (2 hosts in 1.17 seconds).

On average 2 of the 51 machines were banned with 1 additional machine being infected but going off-LAN before the network reached a stable state.

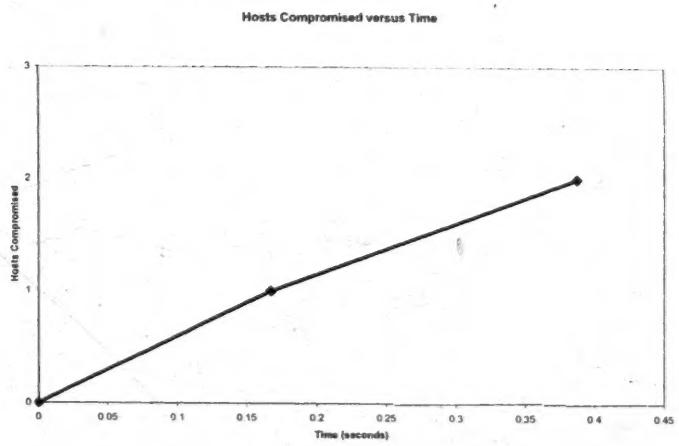

Figure 4.13: Best case result from snort testing with only the custom signature loaded

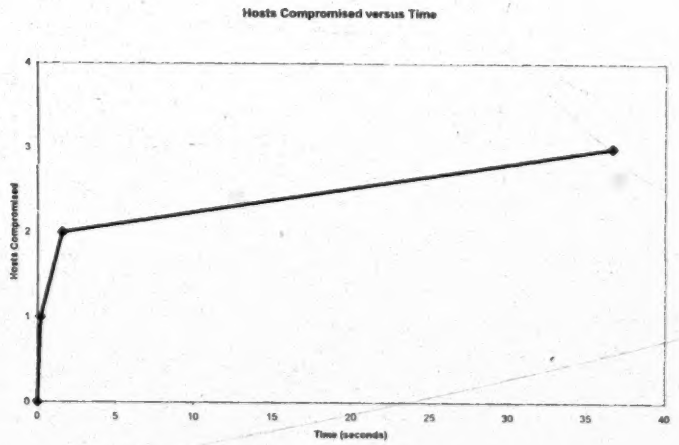

Figure 4.14: Worst case result from snort testing with all signatures + the custom signature loaded

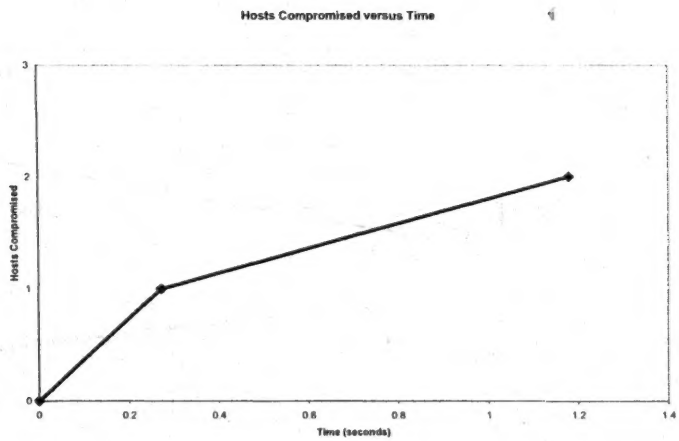

Figure 4.15: Best case result from snort testing with all signatures + the custom signature loaded

From these experiments we can conclude that Snort's signature matching engine is sufficiently fast, and that a purpose-built system to detect and mitigate worms using Snort as the detection mechanism could load the entire Snort signature database with minimal performance degradation. This increases the utility of the detection device as Snort is capable of detecting a number of known attacks, including e-mail borne viruses, network reconnaissance, etc. However, it should be noted that the effectiveness of Snort as a worm detection mechanism relies entirely on the quality of the signatures available. If a signature is poor, or does not exist at all, the Snort device will be unable to properly classify the infected/suspicious traffic thereby allowing said traffic to flow on the network unabated as discussed in Section 3.1.5.

4.8 Honeywall

The base configuration was to setup a honeypot for each of the IPs in 10.0.0.0/23 not registered in the table of used IPs (i.e. for each IP not assigned to a real machine, create a honeypot associated with the unregistered IP). Figure 4.16 illustrates how

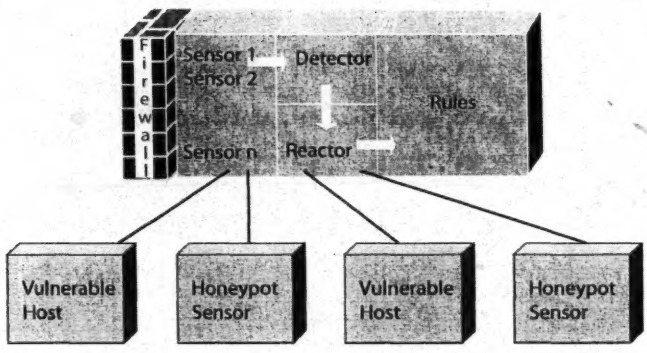

Figure 4.16: Honeywall

honeypot sensors are intermixed throughout the real computer systems on a network. Each honeypot acts as a sensor within the network for detecting unexpected network traffic. Once a sensor has been tripped by unexpected network traffic to the sensor, the honeywall reaction mechanism will add the source IP of the unexpected traffic to a block list, thereby preventing that source IP from communicating with any assets on the network.

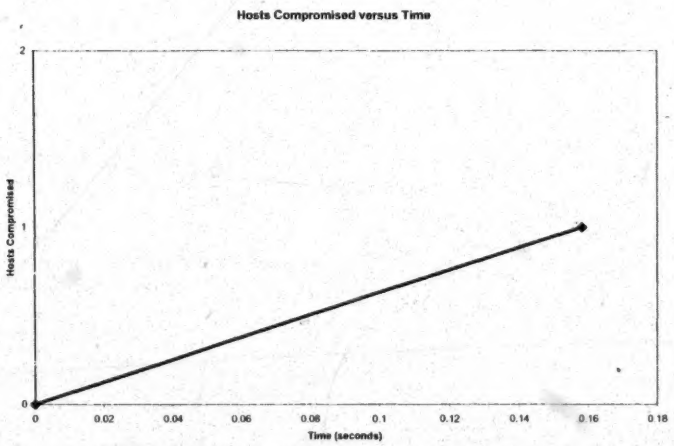

Figure 4.17: Best case result from honeywall testing

Figure 4.18 shows the poorest results observed in the experimental setup (1 host

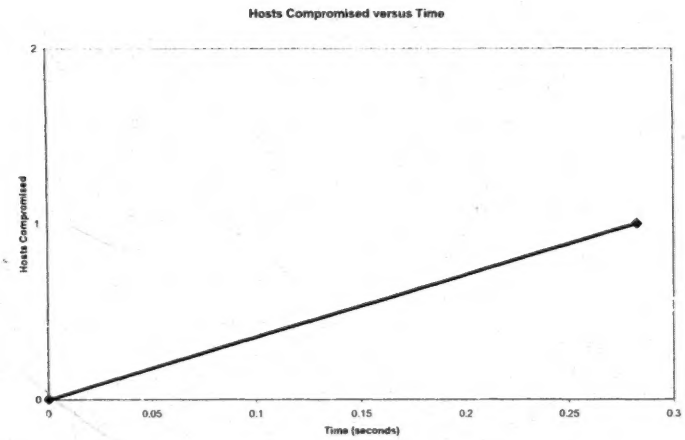

Figure 4.18: Worst case result from honeywall testing

in 0.28264 seconds), while Figure 4.17 shows the best results received (1 host in 0.158041 seconds). In all trials only the initially infected machine was banned – traffic from the initially infected host was contained before it could reach any live machines on the network. In other words, none of the vulnerable machines beyond the initial infected host were compromised.

4.9 Summary of Findings

Throughout the trials the only variable in the experiments was the detection mechanism used within the test environment. Each trial was monitored for the time it took for a detection mechanism to prevent further infection. In addition, once a stable state was reached, a tally of the total number of hosts compromised was taken. Table 4.3 shows the averaged results for all of the trials performed, while Figure 4.19 provides those same results in graph form.

From these results the conclusion can be drawn that using a honeywall to detect and mitigate worms can be an effective mechanism for protecting network assets against threats which can not be dealt with by the other state-of -the-art solutions

Protection Type	Compromised Systems	Time to Stable State (seconds)
None	51	9
Per subnet	27	27
pf (50con / 4sec)	21	77
pf (8 con / 4sec)	2	5.7
pf (28con / 4sec)	11	16
Snort (custom rule)	2	0.78
Snort (all + custom)	2	0.99
Honeywall	1	0.27

Table 4.3: Summary of Results for all Configurations

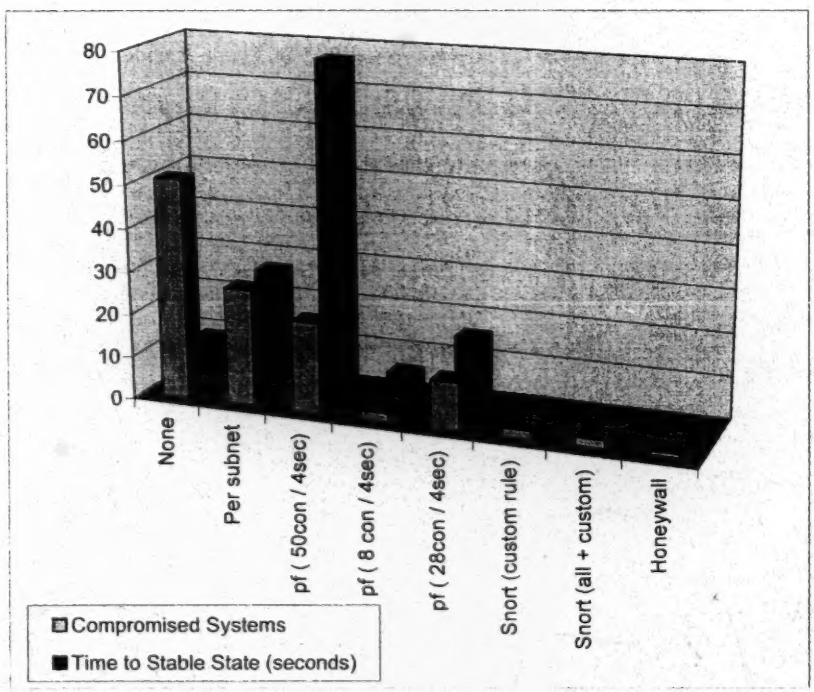

Figure 4.19: Graph of Summary of Results for all Configurations

evaluated in this Chapter. A honeywall does not use signatures to detect worms, which provides an advantage when compared to Snort because a honeywall is capable of protecting against all forms of unexpected traffic and not just those for which a signature has been created. Likewise, the honeywall is more flexible than the threshold model demonstrated with the pf experiments as it does not monitor individuals computer's use of the network. These results are sufficiently promising that further research and development of honeywalls should be made. As will be discussed in Chapter 6, there are numerous areas of future work and improvement to refine the honeywall mechanism so it can be deployed in a real world setting.

The data in this chapter has been presented to the CanSecWest security conference in April 2006 [64].

Chapter 5

Discussion

As a result of its light weight implementation and interaction models the honeywall is ideally suited for deployment at a network layer. By placing the honeywall functionality on the switch it is anticipated that the response latency of isolating the affected systems can be minimized as only the switch itself would be in the decision making process. this is in contrast to the existing implementation where, in effect, there are two systems – the honeywall sensor network, and the reactor – which are involved.

The honeywall as implemented in the testing harness required very few system resources, impacting memory and CPU minimally. On a 100Mbit network it was possible to implement the honeywall using less than one percent of the test system's CPU. As many commodity switch and firewall units utilize a conventional x86 processor within their architecture, it is expected that the honeywall created for this dissertation could be ported and deployed onto these devices with minimal negative impact on their capabilities and available resources. It should be noted that while the honeywall used in this testing was run on x86 architecture, it is anticipated that the architecture and technology could be redeployed on other CPUs.

The true strength of the honeywall is apparent when it is deployed in such a way as to only inspect local network traffic. By limiting the exposure of the honeywall sensors to the local network the possibility of directed attacks from outside sources seeking to exploit the behaviour of the honeywall is reduced.

For a honeywall to be deployed in a production environment the following best

practices are recommended:

1. The border switch/edge network device should have anti-spoofing rules enforced. Anti-spoofing rules prevent a member of the outside network from crafting and injecting traffic that appears to originate from the inside network. If anti-spoofing is not enabled it becomes trivial to turn the honeywall against the inside network as an attacker could craft and send packets to the honeywall that appeared to originate from a real network asset, resulting in the honeywall blocking the real network asset.

2. Exception rules must be added to permit multicast and broadcast traffic as by these transmissions are by nature designed to reach multiple systems on the inside network simultaneously.

3. Network assets that are known to scan a network must be identified and special exceptions must be made for expected traffic from those systems. For example, in the normal operation of a network the LAN administrator may wish to probe the computer systems on the network. Tools such as those described in Section 2.2 could be legitimately used to scan a local network en masse, an action that would undoubtedly touch at least one honeywall sensor. If an exception has not been created for this workstation it will be dropped from the network once traffic originating from it reaches the honeywall's sensors.

Chapter 6

Conclusion and Future Work

This dissertation has provided a background in the current state-of-the-art technologies used in worm detection and mitigation. these technologies have been taken and tested against the honeywall technology developed for this dissertation. The testing performed and following discussions showed the strengths and weaknesses of these solutions, highlighting how the honeywall technology could be successfully used to detect and stop worms on a network before more conventional mechanisms would be able to respond as shown in Figure 4.3.

While the testing performed for this dissertation strongly suggests that honeywalls could be developed into a highly effective device or mechanism to supplement existing IDS technologies, it is only the beginning. In the following sections I outline several promising areas of future work that may be useful in refining the honeywall technology into something that could be widely deployed to great effect.

6.1 False Positive Reduction

An area for research is to investigate if any benefit can be derived by using higher interaction honeypots to replace or supplement the extremely low interaction honeypots used in this implementation. From a real-world standpoint the, low-interaction honeypots are useful because they provide speed and ease of detection when used as sensor nodes for worm propagation. However, the "if it touches a honeypot it must be bad" metric may be too aggressive in networks where broadcast protocols are used, or where client systems scan the entire network. By using higher interaction

honeypots, the tradeoff may be made allowing deeper inspection on the unsolicited traffic seen by the honeypots, thus allowing a more informed decision about the legitimacy of the traffic. However, as the high interaction honeypot model requires more transactional information to take place, it necessarily slows down the isolation phase of the process. While the higher interaction honeypots will most likely slow down the response time of the system, as compared to a system totally composed of ultra-low interaction honeypot sensors, it may be possible to find a balance between the ultra-low and higher interaction honeypots that still provides a sufficiently low response time to block fast spreading worms.

It may be possible to create a more equitable distribution of high and low interaction honeypots through careful routing policies. A known high risk port, such as 445, could be routed to a higher interaction honeypot sensor to determine the validity of the traffic. Similarly, infrequently used ports could be routed to lower interaction honeypot sensors to provide greater speed in response.

Hysteresis could be introduced into the response system by implementing a counter-based honeypot. A threshold for traffic within a fixed period of time could be set, and when that threshold is exceeded the source host will be blocked, not unlike the threshold mechanism described in Section 4.7.1. Therefore the use and placement of high interaction honeypots versus low interaction honeypots should also be looked at.

6.2 Honeypot Sensor Placement

In the experiments reported, uniform placement of honeypots in the empty nodes between the evenly spaced real systems was used. As one scan pattern can't be counted on for mass spreading worms, differing placements of honeypots, as well as investigating the density and clustering/grouping of sensor nodes, may result in better detection and isolation of zero day threats to the network. Ideally the initial infected host system, be it internal or external, will contact a honeypot node before it contacts a real system node.

A further question that comes from the investigation into honeypot placement is

that of the ratio between honeypot sensor nodes and real nodes. In the extreme case of a fully populated local network, no sensor nodes could exist, essentially preventing the honeypot sensor mechanisms from seeing any traffic at all. On the other end of a spectrum, a completely unpopulated network could be fully populated with a set of honeypot sensors, turning every IP address in that subnet into a detector and thus maximizing detection capabilities while completely preventing any real systems from existing on the network. By investigating the ratio of honeypot nodes to real nodes a point of diminishing returns may be found.

6.3 Integration of Solutions

A further hybrid could be created from a combination of the tested setups. For example, Snort used in conjunction with a Honeywall could provide an even better resolution. By leveraging the pattern matching of Snort's known attack database upstream of the Honeywall's unknown threat detection, it is anticipated that both known and unknown attack detection can be improved.

6.4 pf improvements

Since the testing was performed, OpenBSD has progressed from version 3.6 to version 3.9. In OpenBSD 3.8 a new set of features were added to the stateful tracking options in pf called the *overload table* and *flush* [65]. The overload table can be used in conjunction with the existing connection limiting functions in pf to place an offending host into a persistent table that can be further filtered or blocked. Flush complements the overload table by providing an easy mechanism to flush the states of the offending host based on a rule match or by flushing all states for a given host. In the case where Flush is used to block based on a rule match only the traffic matching that rule will be blocked from that host, all other traffic will be allowed. If flush is invoked using the global flag all connections from the offending host will be purged.

The ability to log some or all of the traffic seen also exists in pf. An additional mechanism that may prove useful in further limiting the spread of worms on a

local network proposal follows. If a threshold is met, and the host is added to the overflow table, an additional function call could be made that reconstructs the past n hosts that the blocked machine has communicated with. In the most basic implementation, a decision can be made to block some or all of the traffic from the hosts that were communicated with. It may be possible to improve this banning by association mechanism by blocking the outbound ports/traffic types present on the initial blocked machine.

6.5 Response Times

Further improvements can be made to the response times of the honeywall. Currently a block can go into place in 0.06 seconds when the switch is integrated with the honeywall. As the response time is lowered so too is the window of opportunity for a worm to spread across the network. In our trials the worm payload was approximately 7 kilobytes and we were often able to stop a virus in mid transmission. However, if the payload was smaller or if the honeywall was slower we anticipate that the current revision of software may allow a slightly wider spread infection of the network. By reducing the reaction time this possibility would be eliminated.

A runtime version of Perl was used to perform string matching on the traffic flows through the honeywall aggregator. Improvements could be made to the response time by optimizing how Perl is used. For example, a pre-compiled Perl executable could be created so the code is not interpreted at runtime. Alternately, the string matching could be performed through a mechanism like *PCRE* [66] or using the built in regexp functions in a language such as C++.

Bibliography

[1] David Emm. Bugwatch: The real cost of sobig, 2003. http://www.vnunet.com/news/1143284.

[2] Fred Cohen. Computer viruses - theory and experiments. Technical report, University of Southern California, 1984.

[3] Robert Slade. Apple virus. http://www.cknow.com/vtutor/vtsladeapple.htm, 2001.

[4] elk@cloner. *A (long) story about an (old) Apple 2 virus.* http://www.skrenta.com/cloner/clone-post.html.

[5] Mikko Hypponen. F-secure virus descriptions: Monkey. Technical report, F-Secure, Undated. http://www.f-secure.com/v-descs/monkey.shtml.

[6] English Wikipedia. *Timeline of Linux development*, 2005. http://july.fixedreference.org/en/20040724/wikipedia/Timeline_of_Linux_development.

[7] Sir Timothy Berners-Lee. Longer biography, 2004. http://www.w3.org/People/Berners-Lee/Longer.html.

[8] Nikolair Bezroukov. *Corporate Anti Virus Defense Secrets.* Online Publication, 1997. http://www.softpanorama.org/Antivirus/AV_Secrets/Vgallery/concept.shtml.

[9] CMU. Cert advisory ca-1999-04 melissa macro virus. http://www.cert.org/advisories/CA-1999-04.html.

[10] US Department of Justice. Creator of "melissa" computer virus pleads guilty to state and federal charges. http://www.usdoj.gov/criminal/cybercrime/melissa.html.

[11] David Moore, Vern Paxson, and et al. The spread of the sapphire/slammer worm. Technical report, Cooperative Association for Internet Data Analysis - CAIDA, 2003.

[12] Usenix Security 2002. *Warhol Worms: The Potential for Very Fast Internet Plagues*, 2002. http://www.cs.berkeley.edu/nweaver/warhol.html.

[13] David Moore, Colleen Shannon, and et al. Internet quarantine: Requirements for containing self-propagating code, 2003.

[14] Lance Spitzner. *Honeypots Definitions and Value of Honeypots*, 29 May, 2003. http://www.tracking-hackers.com/papers/honeypots.html.

[15] Michael Lyle. Attacks and countermeasures a study of network attack classes and security components to protect against them. Technical report, Recourse Technologies Inc., 1997.

[16] Fyodor. Remote os detection via tcp/ip stack fingerprinting. Technical report, www.insecure.org, 2002. http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[17] Ofir Arkin and Fyodor Yarochkin. Xprobe v2.0. http://www.sys-security.com, 2002.

[18] Michal Zalewski. *The new p0f: 2.0.5.*, 2005. http://lcamtuf.coredump.cx/p0f.shtml.

[19] Johanns Ulrich. *MyDoom-O hits search engines hard.* http://www.incidents.org/diary.php?date=2004-07-26&isc=3dd1f0e697167ede0f0a3f38cd26340d.

[20] Larry Rogers. What is a distributed denial of service (ddos) attack and what can i do about it? http://www.cert.org/homeusers/ddos.html.

[21] J. Mogul, S. Deering, and et al. Path mtu discovery. Technical report, Stanford University, 1990. ftp://ftp.ietf.org/rfc/rfc1191.txt.

[22] Internet Security Systems. Syn flood. Technical report, Internet Security Systems, March 3, 2006. http://www.iss.net/security_center/advice/Exploits/TCP/SYN_floods/default.htm.

[23] Microsoft. Microsoft security bulletin ms03-026. Technical report, Microsoft Corporation, 2003. http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx.

[24] Microsoft Corporation. *Acknowledgement Policy for Microsoft Security Bulletins.* http://www.microsoft.com/technet/security/bulletin/policy.mspx.

[25] Cisco Systems. *Cisco Product Security Incident Response.* http://www.cisco.com/warp/public/707/sec_incident_response.shtml.

[26] Aleph One. Smashing the stack for fun and profit. *Phrack*, 49(14 of 16), October 1996.

[27] Microsoft Security. Microsoft security bulletin ms04-011. Technical report, Microsoft Corporation, 2004. http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx.

[28] CERT. Cert advisory ca-2003-09 buffer overflow in core microsoft windows dll. Technical report, Carnegie Mellon University, 2003. http://www.cert.org/advisories/CA-2003-09.html.

[29] Takayoshi Nakayama. W32.sasser.g. Technical report, Symantec Corporation, 2004 http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.g.html.

[30] Nicholas J. Puketza, Kui Zhang, and et al. A methodology for testing intrusion detection systems. *IEEE*, page 25, 1996.

[31] SCS International Symposium on Performance Evaluation of Computer and Telecommunication Systems. *A Distributed Concurrent Intrusion Detection Scheme Based on Assertions*, 1999.

[32] Steve Martin. Anti-ids tools and tactics. *SANS Institute Information Security Reading Room*, 2001.

[33] Thomas H. Ptacek. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., 1998.

[34] Paul Helman, Gunar Liepins, and Wynette Richards. Foundations of intrusion detection. In *Proceedings of the Fifth Computer Security Foundations Workshop*, pages 114–120, 1992.

[35] James Cannady. Artificial neural networks for misuse detection. Technical report, School of Computer and Information Sciences, 1998.

[36] Murali Kodialuam and T.V. Lakshman. Detecting network intrusions via sampling: A game theoretic approach, 2003.

[37] Alec Yasinsac. Detecting intrusions in security protocols. In *Proceedings of First Workshop on Intrusion Detection Systems, in the 7th ACM conference on Computer and communications Security, June 2000*, pages 5–8, 2000. citeseer.ist.psu.edu/yasinac00detecting.html.

[38] Thomas Daniels and Eugene Spafford. Identification of host audit data to detect attacks on low-level ip. *Journal of Computer Security*, 7(Issue 1):3–35, 1999.

[39] Fred Cohen. 50 ways to defeat your intrusion detection system, 1997. http://all.net/journal/netsec/1997-12.html.

[40] Niels Provos. *Honeyd Download and Release Information.* http://www.honeyd.org/release.php.

[41] Niels Provos. *Honeypot Background.* http://www.honeyd.org/background.php.

[42] Tony Bradley. *Introduction to Packet Sniffing*, 2005. http://netsecurity.about.com/cs/hackertools/a/aa121403_p.htm.

[43] Ryan Spangler. *Packet Sniffer Detection with AntiSniff.* http://www.packetwatch.net/documents/papers/snifferdetection.pdf.

[44] Stephen Venter. *Ethernet MAC address spoofing in Linux. http://whoozoo.co.uk/mac-spoof-linux.htm*, 2005. http://whoozoo.co.uk/mac-spoof-linux.htm.

[45] Sean Whalen. An introduction to arp spoofing. http://node99.org/projects/arpspoof, April 2001.

[46] Doug Song. *dsniff.* http://naughty.monkey.org/~ dugsong/dsniff.

[47] Richard Silverman. dnsiff and ssh: Reports of my demise are greatly exaggerated. Technical report, O'Reilly, 2000.

[48] Doug Song. *fragrouter – network intrusion detection evasion toolkit*, 2005. http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/fragrouter.html.

[49] Sun Tzu. *The Art of War: The Denma Translation (Shambhala Library).* Shambhala Publications, Inc., 2001.

[50] Mucho Maas and Minor Threat. *Toneloc v1.10*, 1994.

[51] Fyodor. The art of port scanning. *Phrack Magazine*, 7(51), 1997.

[52] Ryan W. Maple. Engarde secure linux security advisory. Technical report, Guardian Digital Inc., 2002. http://www.linuxsecurity.com/advisories/other_advisory-2564.html.

[53] Harald Welte. *netfilter/iptables project homepage - The netfilter.org project*, January 30, 2006. http://www.netfilter.org.

[54] Marty Roesch. *Snort Project web page.* http://www.snort.org.

[55] Anonpoet. *Hogwash Project Homepage*. http://sourceforge.net/projects/hogwash.

[56] Oleg Kolesnikov and Wenke Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic, 2004. http://citeseer.ist.psu.edu/kiolesnikov04advanced.html.

[57] Don Parker. Obfuscated shellcode, the wolf in sheep's clothing (part 2). *WindowSecurity.com*, June 7 2005. http://www.windowsecurity.com/articles/Obfuscated-Shellcode-Part2.html.

[58] Matt Jonkman and James Ashton. *The Bleeding Edge of Snort – Breaking Snort Signatures*, February 27, 2006. http://www.bleedingsnort.com/index.php.

[59] Mark Handley and Vern Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. Technical report, AT&T Center for Internet Research at ICSI, 2001. http://www.icir.org/vern/papers/norm-usenix-sec-01-html/index.html.

[60] Honeynet Project maintainers. The honeynet project. Technical report, www.honeynet.org, 2006.

[61] Joseph Corey. Advanced honey pot identification and exploitation. *Phrack Magazine*, 2004. http://www.phrack.org/fakes/p63/p63-0x09.txt.

[62] Neal Krawetz. Anti-honeypot technology. *IEEE Security & Privacy*, 2(1):76-79, 2004. http://ieeexplore.ieee.org/iel5/8013/28290/01264861.pdf.

[63] Neils Provos. A virtual honeypot framework. Technical report, 13th USENIX Security Symposium, 2004. http://www.honeyd.org/worms.php.

[64] Josh Ryder. Real time threat mitigation techniques. CanSecWest Applied Security Conference, 2006. www.cansecwest.com.

[65] www@openbsd.org. *PF: Packet Filtering*, January 1, 2006. http://www.openbsd.org/faq/pf/filter.html.

[66] Philip Hazel. *PCRE - Perl Compatible Regular Expressions*, January 30, 2006. http://www.pcre.org.